University of Bath

PHD

Adaptive Mesh Methods for Numerical Weather Prediction

Cook, Stephen

Award date:
2016

Awarding institution:
University of Bath

Link to publication

Alternative formats
If you require this document in an alternative format, please contact:
openaccess@bath.ac.uk

Copyright of this thesis rests with the author. Access is subject to the above licence, if given. If no licence is specified above, original content in this thesis is licensed under the terms of the Creative Commons Attribution-NonCommercial 4.0 International (CC BY-NC-ND 4.0) Licence (https://creativecommons.org/licenses/by-nc-nd/4.0/). Any third-party copyright material present remains the property of its respective owner(s) and is licensed under its existing terms.

Take down policy
If you consider content within Bath's Research Portal to be in breach of UK law, please contact: openaccess@bath.ac.uk with the details. Your claim will be investigated and, where appropriate, the item will be removed from public view as soon as possible.

Download date: 05. May. 2023

Adaptive Mesh Methods for Numerical Weather Prediction

submitted by

Stephen P. Cook

for the degree of Doctor of Philosophy

of the

University of Bath

Department of Mathematical Sciences

April 2016

COPYRIGHT

Attention is drawn to the fact that copyright of this thesis rests with the author. A copy of this thesis has been supplied on condition that anyone who consults it is understood to recognise that its copyright rests with the author and that they must not copy it or use material from it except as permitted by law or with the consent of the author.

This thesis may be made available for consultation within the University Library and may be photocopied or lent to other libraries for the purposes of consultation with effect from .....
.....

Signed on behalf of the Faculty of Science .....

Acknowledgements

This work was funded by an EPSRC CASE Award with the Met Office.

This thesis would not have been possible without the support of many people. First of all I'd like to thank my supervisors, Chris Budd and Tom Melvin, for your time, attention, support and patience. I'd like to thank my examiners, Alastair Spence and Mike Baines, for your invaluable feedback, without which my thesis would not be what it is.

Next I'd like to thank the staff at the University of Bath, in particular Adrian Hill, Rob Scheichl, Tony Shardlow and Paul Milewski, and Nigel Wood and the Dymanics team at the Met Office. I shared some fantastic times at conferences with Emily, Phil and Terry, and in Exeter with Abeed, Helen, Sam and Tommaso. At Bath, I enjoyed sharing an office with some great people including Amy, Andrea, Curdin, Jack and Katy. I also enjoyed sharing cake with everyone on Wednesdays, and sharing drinks in the parade with everyone at Drinks in the Parade.

Finally I'd like to give special mention to the support from Jen, Johannes and my parents.

Abstract

This thesis considers one-dimensional moving mesh (MM) methods coupled with semi-Lagrangian (SL) discretisations of partial differential equations (PDEs) for meteorological applications. We analyse a semi-Lagrangian numerical solution to the viscous Burgers' equation when using linear interpolation. This gives expressions for the phase and shape errors of travelling wave solutions which decay slowly with increasing spatial and temporal resolution. These results are verified numerically and demonstrate qualitative agreement for high order interpolants. The semi-Lagrangian discretisation is coupled with a 1D moving mesh, resulting in a moving mesh semi-Lagrangian (MMSL) method. This is compared against two moving mesh Eulerian methods, a two-step remeshing approach, solved with the theta-method, and a coupled moving mesh PDE approach, which is solved using the MATLAB solver ODE45. At each time step of the SL method, the mesh is updated using a curvature based monitor function in order to reduce the interpolation error, and hence numerical viscosity. This MMSL method exhibits good stability properties, and captures the shape and speed of the travelling wave well. A meteorologically based 1D vertical column model is described with its SL solution procedure. Some potential benefits of adaptivity are demonstrated, with static meshes adapted to initial conditions. A moisture species is introduced into the model, although the effects are limited.

Contents

Chapter 1

Introduction

Weather Prediction Accurate weather prediction is an important aspect of the modern world. In centres such as the Met Office, a major aspect of this process is numerical weather prediction (NWP), where the state of the atmosphere is approximated, modelled and predicted on large supercomputers.

Each day Meteorological centres around the world create and distribute weather forecasts. Data collected from satellites, buoys, observation stations, weather balloons, boats and aeroplanes is combined with advanced numerical algorithms in order to deliver weather charts, forecasts and weather warnings. A vital step in the numerical weather prediction process is the dynamical core, where the large scale movements of air masses, energy, and moisture are simulated and predicted. In the dynamical core, the physical laws dictating the interactions between energy, mass and momentum are approximated and used to predict future states of the atmosphere based on the best estimates to the present and previous state of the atmosphere.

The physical laws take the form of partial differential equations expressing the conservation of mass, energy and momentum. These equations are discretised in time and space, and the resulting equations, along with initial conditions, boundary data, and any coupling to other models, such as oceanic forcing, are solved with appropriate algorithms on computers.

A popular technique for discretising these laws is using a Lagrangian frame of reference, whereby the temporal derivatives in the PDEs are represented at idealised reference points which track the movement of the general advective flow, or in the case of the atmosphere, with the wind. This is in contrast to an Eulerian method where time derivatives are considered at a number of fixed points in space, as would be observed by a network of stationary sensors, such as weather stations. Semi-Lagrangian (SL) methods, as considered in this thesis, take aspects of both of these methods,

considering reference points which are advected with the flow, but arrive at, or depart from, specified points in space. The Met Office have been using a semi-implicit semi-Lagrangian (SISL) dynamical core in their Unified Model (UM) for NWP since 2002, and for climate forecasting since 2004 [10].

In order for a forecast to be relevant, these computations must be performed quickly. Therefore we are limited in the discretisation to resolutions where we may not be able to represent the full spectrum of behaviour permitted by the PDEs. This can result in certain weather phenomena not being represented by the model. A potential solution to this is to allow the spatial resolution to dynamically change as such features form and disappear. Such a strategy is known as mesh adaptivity.

Mesh Adaptivity Mesh adaptivity has been successfully applied to many problems in fluid mechanics. There are three main types of mesh adaptivity: mesh refinement, where mesh points are added and removed to achieve the required resolution, is known as h-refinement [1]; p-refinement where the mesh is fixed and the order of the numerical method is increased locally, developed in the context of the finite element method; and r-adaptivity, which is the method we adopt here, where the mesh points move as the numerical solution evolves [23, 5, 4]. It is not uncommon to use a combination of refinement techniques, such as hp-refinement [46] or hr-refinement [25].

The aim of this thesis is to study both semi-Lagrangian methods and mesh adaptivity. We start by looking at an idealised model, Burgers' equation, and finish by considering a more meteorologically motivated model, a vertical column model.

1.1 Outline of the Thesis and Main Results

In Chapter 2 we review the current theory and practice of SISL methods, particularly in the context of NWP. We also discuss the viscous Burgers' equation which will be a key example for much of this thesis.

In Chapter 3 we study the semi-Lagrangian (SL) method applied to travelling wave solutions of Burgers' equation on a fixed mesh. In this chapter we perform a backward error analysis to identify the nature of the equation actually being solved by the SL method. This analysis clearly demonstrates that the SL calculation acts to both broaden the width of the travelling wave and to change its speed. We show that this is critically dependent on the CFL value. The error appears to be dominated by the interpolation errors of SL methods and we explore the effect of different interpolation strategies. Good agreement is found between numerical and asymptotic calculations.

One conclusion from this is that a very small mesh size is required for good speed resolution. This motivates the adaptive mesh methods used later.

In Chapter 4 we outline the theory of one dimensional moving mesh methods based on equidistribution. We conduct a series of numerical experiments with these methods. The methods are then generalised to 1+1 dimensional systems and are used to look at meteorological type flows over orography. The effect of different strategies on mesh quality is discussed.

In Chapter 5 we demonstrate the benefits of using mesh adaptivity for both Eulerian and semi-Lagrangian discretisations for Burgers' equation. In particular, we show that due to the use of interpolation in semi-Lagrangian methods, the application of moving meshes to semi-Lagrangian advection schemes is simple and effective at reducing the errors identified in Chapter 3.

In Chapter 6 we consider a set of meteorological problems to which the methods of the previous chapters can be applied. In particular we consider the vertical column meteorological problem and apply a static SISL method to it. This shows that SISL is effective but that mesh smoothness is vital to achieve an effective resolution. We then consider the application of various adaptive mesh strategies to the resolution and evolution of an inversion layer in the vertical column. The key result is that adaptive meshes based on a-priori grids tend to reduce accuracy, but that adapting to the curvature of the solution gives a much better resolution of the solution. Finally in this chapter we consider the application of SISL type methods to problems with a high moisture content in hopes of including more forcing. Unfortunately this did not behave as expected, but the derivations are included.

Finally in Chapter 7 we draw conclusions from our findings and discuss potential avenues for future work.

Code for some of the algorithms described is included in the Appendix.

Chapter 2

Review of Semi-Lagrangian Methods in Numerical Weather Prediction and Burgers' Equation

2.1 Literature review

A semi-implicit semi-Lagrangian (SISL) scheme is a method for numerically integrating partial differential equations (PDEs) which has found widespread use in numerical weather prediction (NWP) and climate modelling. In terms of modelling fluids, one can think of a Lagrangian scheme as having a frame of reference moving with the fluid flow, as opposed to an Eulerian scheme, which uses a fixed frame of reference. A purely Lagrangian scheme would be equivalent to releasing a number of particles into a fluid and tracking their properties over time. The drawback to this approach is that, after enough time, the "particles" might start to cluster and leave areas devoid of particles, and hence information. A semi-Lagrangian method avoids this potential short-coming by only considering fluid trajectories which arrive at a mesh-point after a short amount of time, typically one or two time steps.

Semi-Lagrangian (SL) schemes usually have small time truncation errors. Furthermore Robert (1981,1982) [42, 43] showed that in combining a SL scheme with a semi-implicit scheme, one can take large time steps and retain good accuracy of the numerical solution.

In this chapter we present some of the history and implementation choices for SISL schemes in NWP, in particular the discretisation in time, discretisation in space, choice of interpolant and general solution procedure. Following this, we give a brief introduction to Burgers' equation, a nonlinear advection PDE which we use as a motivating

example for SL methods and later for moving meshes.

2.2 Time Discretisation

We now look at a few time-discretisations used in semi-Lagrangian schemes: three-time-level, two-time-level and semi-implicit schemes, off-centring, and iterative-implicit schemes.

We are looking to find numerical solutions to advection dominated PDEs with an advective velocity given by $\mathbf{V}$. Such a PDE can be expressed using a Lagrangian derivative of the form

$$\frac{\mathrm{D}F}{\mathrm{D}t} - G(\mathbf{x},t) = R(\mathbf{x},t)\,. \tag{2.1}$$

The terms $G(\mathbf{x},t)$ and $R(\mathbf{x},t)$ represent terms that require implicit and explicit treatment respectively. Here the Lagrangian derivative of F is given by

$$\frac{\mathrm{D}F}{\mathrm{D}t} \equiv \frac{\partial F}{\partial t} + (\mathbf{V}\cdot\nabla)F\,, \tag{2.2}$$

which is the derivative along a fluid trajectory $\vec{\mathbf{x}}(t)$. To find this fluid trajectory we can look at the Lagrangian derivative of $\mathbf{x}$,

$$\begin{aligned}\frac{\mathrm{D}\mathbf{x}}{\mathrm{D}t} &= \frac{\partial \mathbf{x}}{\partial t} + \mathbf{V}\cdot\nabla\mathbf{x} \\ &= \mathbf{V}(\mathbf{x},t)\,.\end{aligned} \tag{2.3}$$

The trajectories $\vec{\mathbf{x}}(t)$ are then solutions to the equation

$$\frac{\mathrm{d}\vec{\mathbf{x}}}{\mathrm{d}t}(t) = \mathbf{V}(\vec{\mathbf{x}}(t),t)\,. \tag{2.4}$$

Equation (2.3) is the *kinematic equation* and will be the subject of Section 2.4.

We will consider solving these problems on a fixed mesh in time and space. In this mesh we have time levels t^n, $n = 0,1,\ldots$, and spatial points x_j, $j = 0,1,\ldots$, with $\Delta t = t^{n+1} - t^n$ and $\Delta x = x_{j+1} - x_j$.

We can integrate equation (2.1) along a trajectory $\vec{\mathbf{x}}(t)$ from t^n to t^{n+1} to give

$$F\left(\vec{\mathbf{x}}(t^{n+1}),t^{n+1}\right) - F\left(\vec{\mathbf{x}}(t^n),t^n\right) = \int_{t^n}^{t^{n+1}} \left[G\left(\vec{\mathbf{x}}(t),t\right) + R\left(\vec{\mathbf{x}}(t),t\right)\right]\mathrm{d}t\,. \tag{2.5}$$

Typically, we consider trajectories which start at time t^{n-1} or t^n and which arrive at known fixed grid points at time t^{n+1}. We call these grid points the *arrival points* of

the trajectories, denoted by

$$\mathbf{x}_A := \vec{\mathbf{x}}(t^{n+1})\,. \tag{2.6}$$

Such trajectories start at points $\vec{\mathbf{x}}(t^{n-1})$ or $\vec{\mathbf{x}}(t^n)$.

Early work on semi-Lagrangian methods [54, 45] was based around three-time-level (3TL) schemes, where the Lagrangian equation (2.5) with $G(\mathbf{x}, t) = 0$ is discretised at time levels t^{n-1}, t^n and t^{n+1} as

$$F(\mathbf{x}_A, t^{n+1}) - F(\mathbf{x}_A - 2\alpha_{\mathrm{x}}, t^{n-1}) = 2\Delta t R(\mathbf{x}_A - \alpha_{\mathrm{x}}, t^n)\,. \tag{2.7}$$

Here $2\alpha_{\mathrm{x}}$ is the displacement over the time step from t^{n-1} to t^{n+1} along a fluid trajectory which arrives at $\mathbf{x}_A$ at time t^{n+1}. It follows that

$$2\alpha_{\mathrm{x}} = \mathbf{x}_A - \vec{\mathbf{x}}(t^{n-1}) \tag{2.8}$$

and

$$\alpha_{\mathrm{x}} \approx \mathbf{x}_A - \vec{\mathbf{x}}(t^n)\,. \tag{2.9}$$

Figure 2-1 shows a 1D trajectory with $\mathbf{x}_A$, $\mathbf{x}_A - \alpha_{\mathrm{x}}$ and $\mathbf{x}_A - 2\alpha_{\mathrm{x}}$. Note that α_{x} is at this stage unknown and will be approximated as part of the solution process. We delay discussion of the calculation of the fluid trajectories $\mathbf{x}(t)$ (and hence α_{x}) from $\mathbf{V}$ until Section 2.4.

A Courant number is a dimensionless parameter which, for a typical speed u, spatial step Δx and time step Δt is given by

$$\nu_{\mathrm{CFL}} = \frac{u\Delta t}{\Delta x}\,. \tag{2.10}$$

The Courant-Friedrichs-Lewy (CFL) condition states that for an explicit finite difference scheme applied to an advection dominated PDE to be stable, we require that $\nu_{\mathrm{CFL}} < 1$. In a typical global circulation model used for NWP, we have a horizontal mesh width of $\Delta x \approx 100\,\mathrm{km}$, vertical mesh width of $\Delta z \approx 100\,\mathrm{m}$, horizontal wind speed typically around $u \approx 100\,\mathrm{m\,s^{-1}}$ and vertical wind speed around $w \approx 0.1\,\mathrm{m\,s^{-1}}$. The CFL condition used when considering horizontal or vertical winds, implies that

$$\frac{u\Delta t}{\Delta x} < 1 \quad \text{and} \quad \frac{w\Delta t}{\Delta z} < 1 \tag{2.11}$$

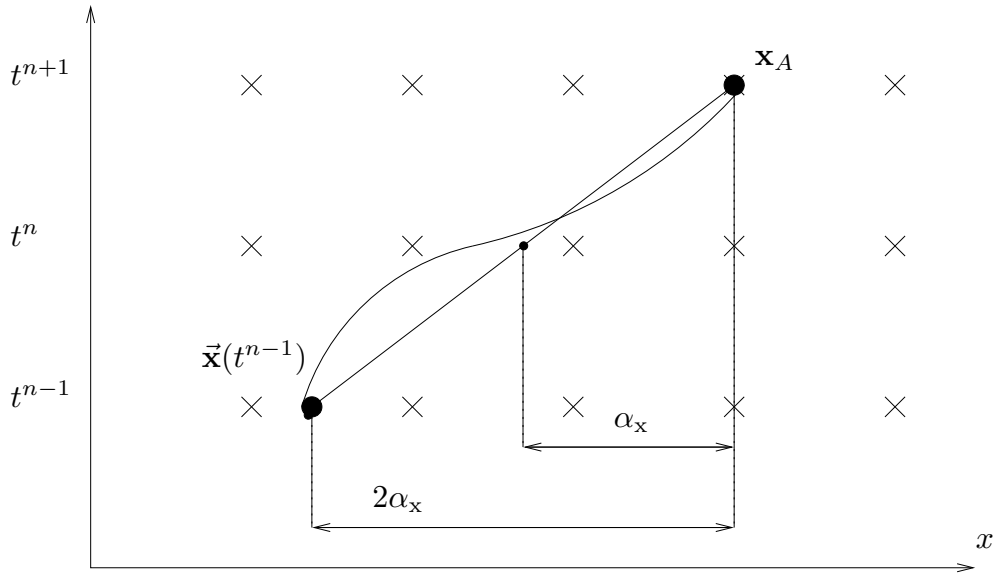

Figure 2-1: Example of arrival and departure points for a 3TL scheme in 1D. For equations (2.7) and (2.15), the terms in R require interpolating at $\mathbf{x}_A - \alpha_{\mathrm{x}}$ and the terms in F and G at $\mathbf{x}_A - 2\alpha_{\mathrm{x}}$.

respectively. Both of these conditions give an approximate restriction on the time step as

$$\Delta t < 1000\,\mathrm{s}\,. \tag{2.12}$$

Acoustic waves have a typical speed of $u_s \approx w_s \approx 340\,\mathrm{m\,s^{-1}}$. For horizontal acoustic waves, the CFL condition gives

$$\Delta t < \frac{1000}{3.4}\,\mathrm{s}\,, \tag{2.13}$$

but for vertical acoustic waves, we have

$$\Delta t < \frac{1}{3.4}\,\mathrm{s}\,, \tag{2.14}$$

a severe restriction on the time step.

Robert [42, 43] presented the semi-implicit semi-Lagrangian (SISL) scheme, whereby fast terms (such as acoustic and gravity waves) are treated implicitly, while slower terms (such as the advective terms) can be treated explicitly, improving the stability of semi-Lagrangian methods.

If we collect the terms allowing for an explicit treatment into $R(\mathbf{x}, t)$ and the terms requiring an implicit treatment into $G(\mathbf{x}, t)$, then a 3TL SISL discretisation of equa-

tion (2.1) is

$$F(\mathbf{x}_A, t^{n+1}) - F(\mathbf{x}_A - 2\alpha_{\mathrm{x}}, t^{n-1}) - 2\Delta t \left(\theta_u G(\mathbf{x}, t^{n+1}) + (1-\theta_u) G(\mathbf{x}, t^{n-1})\right) = 2\Delta t R(\mathbf{x}_A - \alpha_{\mathrm{x}}, t^n)\,, \quad (2.15)$$

where θ_u is the semi-implicit time-weighting coefficient.

These ideas were later successfully extended to give two-time-level (2TL) schemes [52],[49, §2c] taking the form

$$F(\mathbf{x}_A, t^{n+1}) - F(\mathbf{X}_D, t^n) - \Delta t \left(\theta_u G(\mathbf{x}_A, t^{n+1}) + (1-\theta_u) G(\mathbf{X}_D, t^n)\right) = \Delta t R\left(\frac{\mathbf{x}_A + \mathbf{X}_D}{2}, t^{n+1/2}\right)\,. \quad (2.16)$$

Here,

$$\alpha_{\mathrm{x}} = \mathbf{x}_A - \mathbf{X}_D\,, \quad (2.17)$$

where $\mathbf{X}_D$ is the numerical approximation to the *departure point*,

$$\mathbf{X}_D \approx \vec{\mathbf{x}}(t^n) \quad (2.18)$$

where $\vec{\mathbf{x}}(t^n)$ corresponds to the trajectory that arrives at the *arrival point* $\mathbf{x}_A = \vec{\mathbf{x}}(t^{n+1})$. A 2TL trajectory is shown in Figure 2-2 with an arrival point $\mathbf{x}_A$ and departure point $\mathbf{X}_D$.

A 2TL scheme has the potential advantage of being twice as fast as the 3TL scheme, using double the time step of a corresponding 3TL scheme. However, an important factor of such a 2TL scheme is making sure that the trajectory computations are second-order accurate in time [52].

When the semi-implicit weighting coefficient is taken as $\theta_u = \frac{1}{2}$, the 2TL formulation (2.16) gives an implicit Crank-Nicholson integration of $G(\mathbf{x}(t), t)$ with respect to t.

Rivest, Staniforth and Robert (1994) [41] showed that, in the presence of orography, the choice of $\theta_u = \frac{1}{2}$, i.e. a centred explicit time-weighting, leads to *spurious resonance*, non-physical numerical waves which grow in time. The solution to this is to off-centre the implicit time-weighting, and to take $\theta_u > \frac{1}{2}$, with the authors suggesting a value of $\theta_u = 0.7$.

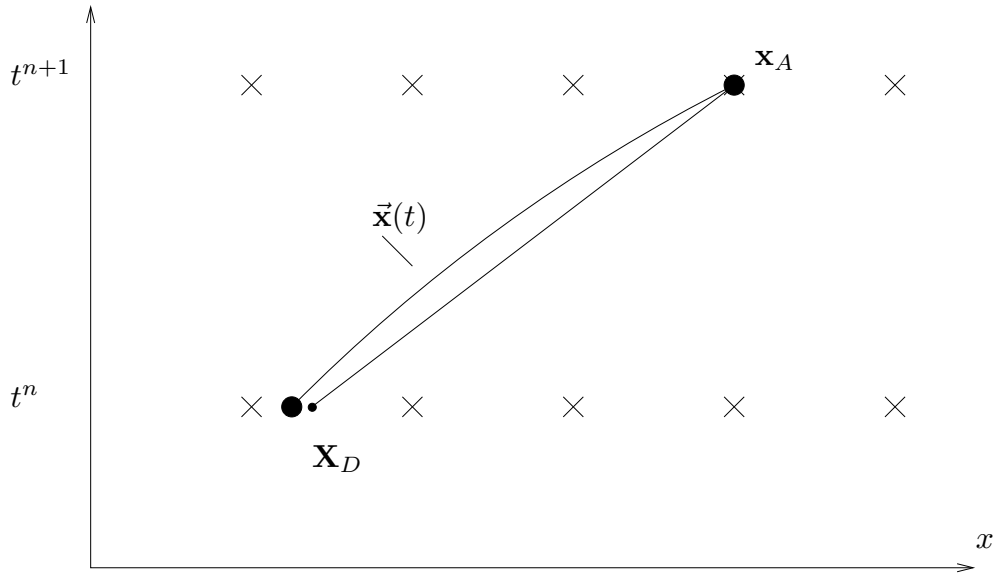

Figure 2-2: Example of arrival and departure points for a 2TL scheme in 1D. For equation (2.16), the terms in F and G are interpolated at $\mathbf{X}_D$, the numerical approximation to $\vec{\mathbf{x}}(t^n)$, and the terms of R, if used, are interpolated at the midpoints $\frac{1}{2}(\mathbf{x}_A + \mathbf{X}_D)$.

In formulating the Canadian Climate Model, Côté et al. (1998) [8] use a 2TL scheme, treating all terms in a time-centred way. This results in all terms being included in G, hence $R = 0$, and we can rearrange equation (2.16) as

$$F_A^{n+1} - \theta_u \Delta t G_A^{n+1} = F_D^n - (1 - \theta_u)\Delta t G_D^n \,, \tag{2.19}$$

where subscripts A and D denote evaluation at $\mathbf{x}_A$ and $\mathbf{X}_D$ respectively, and superscript n and $n + 1$ denote evaluation at time t^n and t^{n+1} respectively. This is solved by linearising the fields about static states and iterating, giving an iterative-implicit semi-Lagrangian scheme.

Finally, the Met Office's dynamical core uses an iterative-implicit scheme, though some terms are iteration lagged [56, §5.1], where certain terms in G are updated less frequently. This reuse of such terms in G leads to computationally cheaper code. A 1D example using such a scheme, showing the linearisation and an iterative solution procedure will be presented in Chapter 6.

We have presented a brief review of the development of SL methods, with some options available when discretising PDEs with advection in time.

Next we briefly look at some of the options available for discretising any spatial derivatives in the expression for G in equations (2.15) and (2.19).

2.3 Spatial Discretisation and Resulting Equation

For problems in NWP, G and R contain terms involving derivatives, so we need a method for calculating these numerically. There are a number of options for spatial discretisation. In this thesis we use finite differences which is the principal method used for approximating derivatives in ENDGame [56], which is the numerical scheme employed by the Met Office to simulate the large-scale movements of air masses in our atmosphere, their *dynamical core*. Finite differences are described in [33] and implemented here in Sections 3.3 and 5.2 with the truncation error for non-uniform finite differences in Section 5.2.1.

In finite volume (FV) methods a differential equation is expressed as an integral equation, and the numerical method is derived by considering fluxes at the boundaries of cells. Semi-Lagrangian FV methods are sometimes called cell-integrated SL methods (CISL) [26, 27]. Finite volume methods are described in [30] for general hyperbolic problems and in relation to meteorology with SL advection in [31]. In ENDGame the continuity equation uses a finite volume discretisation in order to conserve mass [60].

In finite element methods (FEM) and spectral element methods (SEM) space is discretised by a set of approximating functions. The problem is then formulated as finding the best solution in that approximation space.

For FEM the approximating functions are defined locally. Examples of SL methods with FEM can be seen in [28] and [14]. In SEM the approximating functions are orthogonal and usually global [12, §6]. The European Centre for Medium-Ranged Weather Forecasts (ECMWF) employ a 2TL spectral method for their global model [51].

2.4 Departure Point Calculation

An important aspect of semi-Lagrangian schemes is accurately calculating the trajectories which arrive at grid-points, in order to determine where to interpolate the fields. In all but the simplest of cases, the advection will be time dependent, and will often depend on the solution of the PDE. To determine the trajectory, we need to solve the kinematic equation (2.3),

$$\frac{\mathrm{D}\mathbf{x}}{\mathrm{D}t} = \mathbf{V}(\mathbf{x}, t)\,. \tag{2.20}$$

For 3TL schemes, we can integrate this with $\mathbf{x}_A = \vec{\mathbf{x}}(t^{n+1})$ to determine the departure points $\vec{\mathbf{x}}(t^{n-1})$,

$$\mathbf{x}_A - \vec{\mathbf{x}}(t^{n-1}) = \int_{t^{n-1}}^{t^{n+1}} \mathbf{V}(\vec{\mathbf{x}}(t), t)\mathrm{d}t\,. \tag{2.21}$$

The common choice for approximating this integral, solving for $\mathbf{X}(t^{n-1})$, a numerical approximation to $\vec{\mathbf{x}}(t^{n-1})$, is the implicit midpoint rule [49],

$$\mathbf{X}(t^{n-1}) := \mathbf{x}_A - 2\Delta t\mathbf{V}\left(\frac{\mathbf{x}_A + \mathbf{X}(t^{n-1})}{2}, t^n\right), \tag{2.22}$$

This formulation is explicit in time, but implicit in space. The usual approach [49, (5)] is to iterate this equation as

$$\mathbf{X}(t^{n-1})^{\{k+1\}} := \mathbf{x}_A - 2\Delta t\mathbf{V}\left(\frac{\mathbf{x}_A + \mathbf{X}(t^{n-1})^{\{k\}}}{2}, t^n\right), \tag{2.23}$$

where k is the iteration index, in order to find an approximation to the final departure point. This method requires a starting guess of the departure points $\mathbf{X}(t^{n-1})^{\{0\}}$, which we can take as

$$\mathbf{X}(t^{n-1})^{\{0\}} = \mathbf{x}_A - 2\Delta t\mathbf{V}(\mathbf{x}_A, t^n)\,. \tag{2.24}$$

For 2TL schemes, an equivalent approach, and that used by [52, 49], is

$$\mathbf{X}_D = \mathbf{x}_A - \Delta t\mathbf{V}\left(\frac{\mathbf{x}_A + \mathbf{X}_D}{2}, t^{n+1/2}\right), \tag{2.25}$$

where $\mathbf{X}_D$ is a numerical approximation to $\vec{\mathbf{x}}(t^n)$, again iterating to find $\mathbf{X}_D$ as in equation (2.23). There is a slight complication in finding an expression for $\mathbf{V}(\mathbf{x}, t^{n+1/2})$, since we don't readily have $\mathbf{V}$ here. In [49] and references within, they use a time extrapolation to get

$$\tilde{\mathbf{v}}(\mathbf{x}, t^{n+1/2}) = \frac{3}{2}\mathbf{V}(\mathbf{x}, t^n) - \frac{1}{2}\mathbf{V}(\mathbf{x}, t^{n-1})\,, \tag{2.26}$$

then

$$\mathbf{x}_A - \mathbf{X}_D = \Delta t\tilde{\mathbf{v}}\left(\frac{\mathbf{x}_A + \mathbf{X}_D}{2}, t^{n+1/2}\right). \tag{2.27}$$

In [27], the authors use a similar process, but include acceleration terms when determining the extrapolated winds $\tilde{\mathbf{v}}(\mathbf{x}, t^{n+1/2})$.

Following the work of [59] where the solutions were iterated (described above),

Cullen [9] proposed also iterating the departure points. The extrapolation to the half-time level can then be replaced with a time interpolation, namely

$$\tilde{\mathbf{v}}(\mathbf{x}, t^{n+1/2}) = \frac{1}{2}\left(\mathbf{V}(\mathbf{x}, t^n) + \mathbf{V}(\mathbf{x}, t^{n+1})\right) , \tag{2.28}$$

$$\mathbf{x}_A - \mathbf{X}_D = \Delta t \tilde{\mathbf{v}}\left(\frac{\mathbf{x}_A + \mathbf{X}_D}{2}, t^{n+1/2}\right) . \tag{2.29}$$

This was later shown to be beneficial, since using extrapolated winds for the departure point calculation was shown to lead to instabilities in a vertical column model by Cordero, Wood and Staniforth (2005) [7].

Cullen (2001) [9] uses a trapezium-rule discretisation of the kinematic equation (2.20),

$$\mathbf{x}_A - \mathbf{X}_D = \frac{\Delta t}{2}\left(\mathbf{V}(\mathbf{x}_A, t^{n+1}) + \mathbf{V}(\mathbf{X}_D, t^n)\right) . \tag{2.30}$$

Wood, White and Staniforth (2010) [58, §3] compare this 'doubly implicit' discretisation (2.30) with the midpoint discretisation (2.29), and suggest the former is preferable for a number of reasons, including good stability properties and exhibiting improved results in trials. This is what we will use in this thesis.

From this point onwards we opt to use a semi-Lagrangian method which closely mirrors that used by the Met Office in the ENDGame dynamical core [57]. It will be a two-time-level, off-centred, iterated-implicit semi-Lagrangian scheme. The departure points will be calculated with a centred doubly implicit trapezium rule calculation, and all spatial derivatives will be computed with finite differences. We will apply this scheme to Burgers' equation in Chapter 3 and to a vertical column model in Chapter 6. In Figure 2-3 we present a pseudo-algorithm which describes the solution procedure we shall adopt here, loosely mirroring the implementation of ENDGame. It describes 3 loops, a time loop, an outer loop (iterating the departure points) and an inner loop (iterations for the solutions for a fixed departure point).

2.5 Interpolation

It is recognised that a key aspect of the SISL method is the interpolation of the data at the departure point $\mathbf{X}_D$ [47]. There are many different approaches, and we present and review some that have found favour in NWP. In Chapter 3 we will make a detailed analysis of this error for linear interpolation.

First we present a number of techniques for constructing polynomial interpolants.

1: **for** $n := 1$ **to** $n_{\max}$ (*Time loop*) **do**
2: Make an initial estimate of $\mathbf{X}_D$ from $\mathbf{V}^n$
3: **for** $k := 1$ **to** K (*Outer loop*) **do**
4: Interpolate F^n and G^n onto $\mathbf{X}_D$
5: Iteratively solve

$$F^{n+1} - \theta_u \Delta t G^{n+1} = F_D^n + (1 - \theta_u)\Delta t G_D^n \qquad (2.19')$$

for F^{n+1} and G^{n+1} (*Inner loop*)
6: Use new $\mathbf{V}^{n+1}$ to iteratively solve

$$\mathbf{X}_D = \mathbf{x}_A - \frac{\Delta t}{2}\left(\mathbf{V}^{n+1} + \mathbf{V}_D^n\right) \qquad (2.30')$$

for $\mathbf{X}_D$
7: **end for**
8: **end for**

Figure 2-3: Solution procedure for a semi-Lagrangian method as implemented by the Met Office.

Then we discuss monotonicity and a few strategies for enforcing local monotonicity on an interpolant. Finally we talk about other issues, such as the interpolation error and treatment at the boundaries.

2.5.1 Lagrange Interpolants

We now present some of the methods available for forming polynomial interpolants, in particular cubic interpolants. As we shall see, the order of the leading error term in polynomial interpolants of an odd degree interpolant is even, whilst the order of the leading error term for even degree polynomial interpolants is odd. Odd order errors lead do a dispersive error, whilst even order errors lead to diffusive errors, which is preferable for numerical simulations [29, §11.1.1]. Linear interpolation is considered to generate too much diffusion [12, §7.1.1.3], hence many of these methods will be based around cubic interpolants.

It is well known that for $m + 1$ distinct points $z_0, z_1, \ldots, z_m$ and $m + 1$ values $w_0, w_1, \ldots, w_m$, there is a unique degree m polynomial $p_m(z)$ with

$$p_m(z_i) = w_i\,, \qquad i = 0, 1, \ldots, m\,. \qquad (2.31)$$

A simple method for calculating such a polynomial is given by the Lagrange inter-

polation formula [11, §2.5]

$$\pi_i(z) = \prod_{\substack{j=0 \\ i \neq j}}^{m} \frac{z - z_j}{z_i - z_j} \,. \tag{2.32}$$

It is easy to verify that

$$p_m(z) = \sum_{i=0}^{m} \pi_i(z) w_i \tag{2.33}$$

satisfies equation (2.31) at z_i.

We assume that we are approximating a function $f(z)$ with a Lagrange interpolant. The Cauchy Remainder Theorem for polynomial interpolation [11, §3] tells us that for an interpolating polynomial of degree m defined by points $z_0, \ldots, z_m$ and data from a function $f(z)$, the error is given by

$$f(z) - p_m(z) = \frac{(z - z_0)(z - z_1) \cdots (z - z_m)}{(m+1)!} f^{(m+1)}(\xi) \,, \tag{2.34}$$

for $\min_{i=0,\ldots,m}(z_i) < \xi < \max_{i=0,\ldots,m}(z_i)$, where $f^{(m+1)}(z)$ is the $m+1$-th derivative of $f(z)$. For example, the error for the piecewise linear interpolant on $[z_i, z_{i+1}]$ is

$$f(z) - p_1(z) = \frac{(z - z_i)(z - z_{i+1})}{2} f''(\xi) \,. \tag{2.35}$$

When approximating a function $f(z)$ with a piecewise linear interpolant, we can reduce the interpolation error by reducing the grid spacing $z_{i+1} - z_i$ in regions where the second derivative is high. This partially motivates our use of moving meshes in Chapter 5, where we aim to reduce the interpolation error in a SL scheme with linear interpolation.

Another useful way of constructing an interpolating polynomial is with the Newton formula [11, §2.6],

$$p_m(z) = [w_0] + \sum_{k=1}^{m} [w_0, w_1, \ldots, w_k](z - z_0)(z - z_1) \cdots (z - z_{k-1}) \,, \tag{2.36}$$

making use of the divided difference notation

$$\begin{aligned} [w_i, w_{i+1}, \ldots, w_{i+j}] &:= \frac{[w_{i+1}, w_{i+2}, \ldots, w_{i+j}] - [w_i, w_{i+1}, \ldots, w_{i+j-1}]}{z_{i+j} - z_i} \,, \\ [w_i] &= w_i \,. \end{aligned} \tag{2.37}$$

We can use cubic Lagrange interpolants to interpolate over $m > 4$ points by simply

taking the cubic Lagrange interpolant constructed by the 4 points surrounding each interval. This gives a piecewise cubic and globally continuous interpolant which is cheap to compute and evaluate. Near the boundaries we must either take the 4 nearest points, or reduce the order of the interpolant.

2.5.2 Essentially Non-Oscillatory Interpolation

A disadvantage of polynomial interpolation is that it can lead to oscillatory interpolants. This will be addressed below in the context of monotonic cubic interpolants, but we can reduce the effect by being more selective of which points we use to determine the interpolating polynomial. Such a selection process can be implemented via *essentially non-oscillatory* (ENO) interpolation [17], described in Smith (2000) [47, §4.1]. The process for ENO interpolation is relatively simple. We start with the linear interpolant, given in the form of its Newton interpolating polynomial,

$$p(z) = [w_i] + [w_i, w_{i+1}](z - z_i)\,. \tag{2.38}$$

Next we compare the divided difference for the points to either side, then form the quadratic Newton interpolant for the point which gave the lowest divided difference. If the current points are z_i and z_{i+1}, we denote

$$\begin{aligned} z_- &:= z_{i-1}\,, \qquad & w_- &:= w_{i-1}\,, \\ z_+ &:= z_{i+2}\,, \qquad & w_+ &:= w_{i+2}\,, \end{aligned} \tag{2.39}$$

then the new point and value is given by

$$(z_*, w_*) := \begin{cases} (z_+, w_+) & \text{if} \quad |[w_i, w_{i+1}, w_+]| < |[w_i, w_{i+1}, w_-]|\,, \\ (z_-, w_-) & \text{otherwise}\,, \end{cases} \tag{2.40}$$

and we form the ENO quadratic interpolant as

$$p(z) = [w_i] + [w_i, w_{i+1}](z - z_i) + [w_i, w_{i+1}, w_*](z - z_i)(z - z_{i+1})\,. \tag{2.41}$$

We then repeat this process, comparing the two new surrounding points: if in forming the quadratic interpolant we used (z_{i+2}, w_{i+2}), then (z_+, w_+) becomes (z_{i+3}, w_{i+3}); otherwise, we used (z_{i-1}, w_{i-1}) and so the new (z_-, w_-) becomes (z_{i-2}, w_{i-2}). The

new point will then be

$$(z_{**}, w_{**}) := \begin{cases} (z_+, w_+) & \text{if} \quad |[w_i, w_{i+1}, w_*, w_+]| < |[w_i, w_{i+1}, w_*, w_-]| \,, \\ (z_-, w_-) & \text{otherwise}\,, \end{cases} \tag{2.42}$$

and the cubic ENO interpolant is given by

$$\begin{aligned} p(z) = [w_i] + [w_i, w_{i+1}](z - z_i) + [w_i, w_{i+1}, w_*](z - z_i)(z - z_{i+1}) \\ + [w_i, w_{i+1}, w_*, w_{**}](z - z_i)(z - z_{i+1})(z - z_*)\,. \end{aligned} \tag{2.43}$$

A graphical example is given in Figure 2-4, showing the ENO process for an interval $[1, 2]$. As described above, first $[f(1), f(2), f(z_-)]$ is compared with $[f(1), f(2), f(z_+)]$. In this example,

$$|[f(1), f(2), f(3)]| < |[f(1), f(2), f(0)]| \,, \tag{2.44}$$

so the quadratic ENO interpolant is given by the values of f at $\{1, 2, 3\}$: Next, it can be shown that

$$|[f(1), f(2), f(3), f(4)]| < |[f(1), f(2), f(3), f(0)]| \,, \tag{2.45}$$

and so the cubic ENO interpolant is given by the values of f at $\{1, 2, 3, 4\}$.

2.5.3 Cubic Hermite and Spline Interpolation

A cubic Hermite interpolant can be used to specify the values and first derivatives at both ends of an interval. For values w_i and w_{i+1}, and derivatives w_i' and w_{i+1}', the cubic Hermite interpolant over $[z_i, z_{i+1}]$ is given by

$$p(z) = w_i H_1(z) + w_{i+1} H_2(z) + w_i' H_3(z) + w_{i+1}' H_4(z)\,, \tag{2.46}$$

with the four cubic Hermite basis functions (shown in Figure 2-5) given by

$$\begin{aligned} H_1(z) &= \frac{(z_{i+1} - z)^2}{(z_{i+1} - z_i)^2} + \frac{2(z - z_i)(z_{i+1} - z)^2}{(z_{i+1} - z_i)^3}\,, \\ H_2(z) &= \frac{(z - z_i)^2}{(z_{i+1} - z_i)^2} + \frac{2(z_{i+1} - z)(z - z_i)^2}{(z_{i+1} - z_i)^3}\,, \\ H_3(z) &= \frac{(z - z_i)(z_{i+1} - z)^2}{(z_{i+1} - z_i)^2}\,, \\ H_4(z) &= -\frac{(z - z_i)^2(z_{i+1} - z)}{(z_{i+1} - z_i)^2}\,. \end{aligned} \tag{2.47}$$

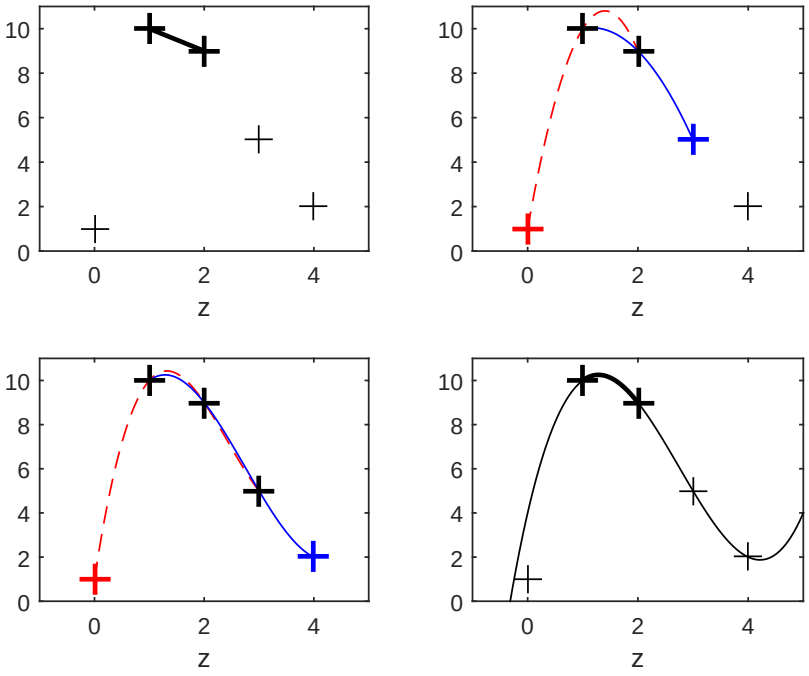

Figure 2-4: Demonstration of ENO interpolation for the interval $[1, 2]$: *(top left)* Linear interpolant; *(top right)* Comparison of quadratic interpolants for $\{0, 1, 2\}$ and $\{1, 2, 3\}$. (*bottom left*) Comparison of cubic interpolants for $\{0, 1, 2, 3\}$ and $\{1, 2, 3, 4\}$. (*bottom right*) Resulting cubic ENO interpolant over $[1, 2]$.

With a method for estimating the gradient at every point, we can construct cubic

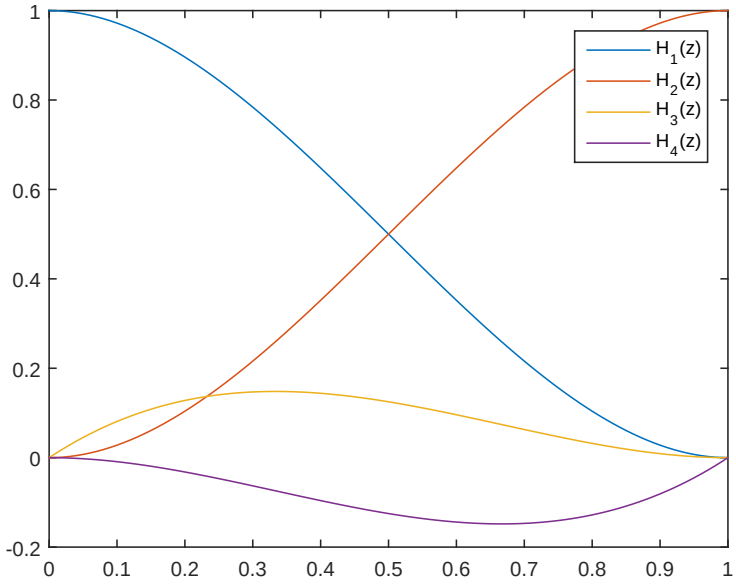

Figure 2-5: The cubic Hermite basis functions (2.47) defined over $z = [0, 1]$.

Hermite interpolants on each interval, giving a globally smooth interpolant.

Cubic spline interpolation [35, §3.3] involves forming piecewise cubic polynomials, chosen such that the first and second derivatives are continuous at all points, and at the end points either the second derivatives are zero (natural cubic spline) or the first derivatives are prescribed (clamped spline).

Most of the methods discussed in [49] use some form of Lagrange interpolation, mostly cubic Lagrange interpolation depending on the 4 points surrounding the departure point. The authors conclude from careful numerical experiments that linear interpolation leads to an unacceptable amount of damping, and cubic Lagrange has a good trade-off of accuracy and computational cost. Linear interpolation is found to be sufficient when determining the departure points.

Non-interpolating schemes as demonstrated by [39] are briefly discussed in [49], where the advection is separated into two parts: a trajectory from the grid point nearest to the departure point, and a trajectory between this grid point and the departure point. The nearest grid point is then used as a shifted frame of reference for an Eulerian scheme with a guaranteed Courant number less than unity.

2.5.4 Monotonicity

We now address the issue of monotonicity and interpolation limiters.

Polynomial interpolation procedures of order higher than 1 are prone to develop oscillations near to sharp jumps (or *low-smoothness points*) called Gibbs phenomenon. We see an example of oscillations near a low-smoothness point in Figure 2-6, where we see a 7-th degree polynomial approximating a step function.

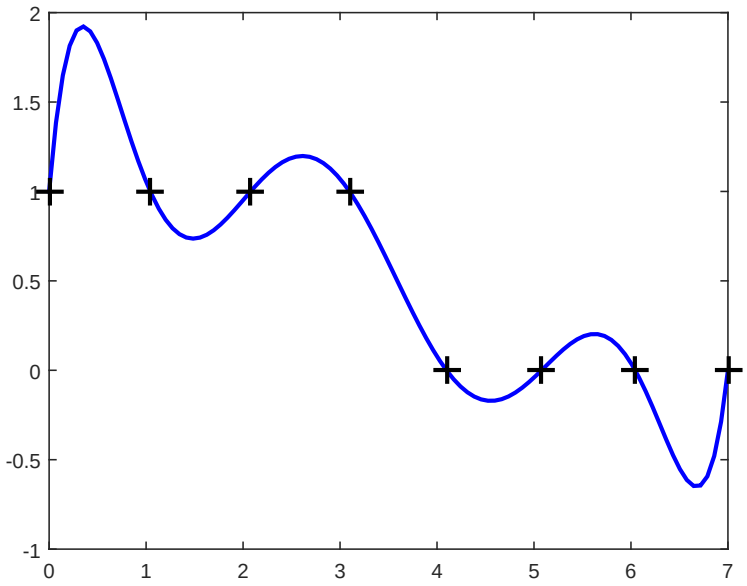

Figure 2-6: Oscillations of a 7-th degree polynomial being used to represent a low-smoothness point with a non-uniform mesh.

When interpolating data associated with physical quantities, there may be situations where the oscillatory behaviour can lead to interpolants which violate physical restrictions, such as negative densities or masses. One solution to this is to ensure local monotonicity, so there are no turning points within any interval (z_i, z_{i+1}), and no new extrema are created.

Monotonic interpolants are discussed with general application in atmospheric fluid dynamics in [48], then for SISL schemes in [2], [55] and [37].

Fritsch and Carlson [15] derived the conditions on the gradients such that a cubic interpolant is monotonic on a given interval. Hyman [22] presents an algorithm which, given points-values and gradients, alters them to give a piecewise monotonic interpolant

where the data are monotonic.

For local monotonicity of non-monotonic data with extrema z_i, such that

$$(f_{i+1} - f_i)(f_i - f_{i-1}) < 0\,,$$

the only choice of gradient which preserves piecewise monotonicity is $f_i' = 0$, i.e. all turning points coincide with a grid-point. Hyman removed the condition of local monotonicity in the intervals to either side of these turning points in the data, leading to better looking interpolants. Such an interpolant is no longer monotonic in the presence of extrema in the data, but no new extrema can be created.

Bermejo and Staniforth [2] present an alternative monotonic scheme which can be applied to any interpolation scheme, whereby the values of the interpolant are restricted to the range of the values at the surrounding grid-points. If we approximate a function $f(z)$ by an interpolating polynomial $p_m(z)$, then a monotonic interpolant for $z \in [z_i, z_{i+1}]$ is easily implemented as

$$p_{m\lim}(z) = \begin{cases} f_- & p_m(z) < f_-\,, \\ f_+ & p_m(z) > f_+\,, \\ p_m(z) & f_- \le p_m(z) \le f_+\,, \end{cases} \tag{2.48}$$

where

$$f_- = \min(f(z_i), f(z_{i+1})) \tag{2.49}$$

and

$$f_+ = \max(f(z_i), f(z_{i+1}))\,, \tag{2.50}$$

or more compactly

$$p_{m\lim}(z) = \max\Big\{ \min(f(z_i), f(z_{i+1})), \min\big[\max(f(z_i), f(z_{i+1})), p_m(z)\big]\Big\}\,. \tag{2.51}$$

This process sacrifices smoothness for monotonicity, but has the advantage of being easy to implement for any interpolation scheme. This is the scheme described in Section 2 of Bermejo and Staniforth [2]: a more thorough description is given in the Appendix of [2] which has been extended to conserve mass when possible [36]. A 1D example using this method is shown in Figure 2-7. We refer to this process as *flux-limited interpolation*, or using a *flux limiter*.

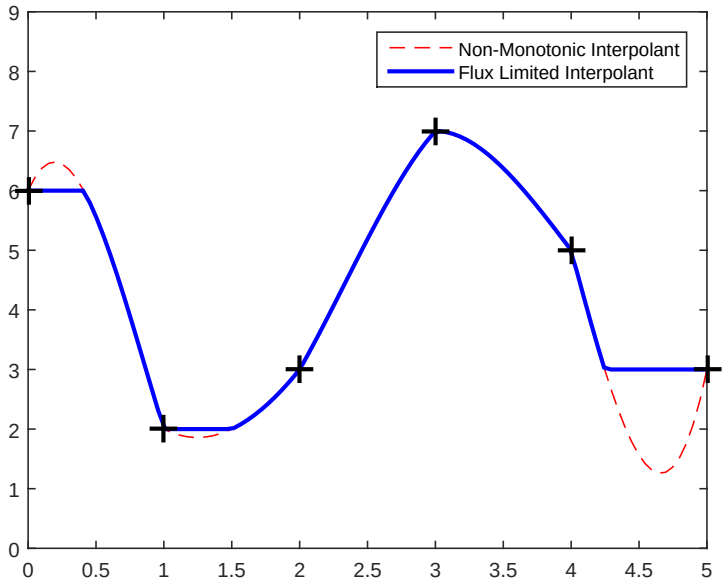

Figure 2-7: Example of a 1D piecewise cubic interpolant with a flux limiter applied. This is a simple method for ensuring local monotonicity.

Williamson and Rasch looked looked at a number of monotonic interpolants and applied them to meteorologically motivated test problems in 1D [37] and 2D [55]. They found that cubic Hermite interpolation with the derivative restrictions proposed by Hyman gave the best results, especially when smoothness was not enforced at the points z_i.

2.5.5 Higher Dimensional Interpolation

While higher dimensional interpolants exist, we can successively use the 1D interpolants described above in an interpolation cascade. This is described in [55, §3c] as a tensor product approach and is demonstrated in a 2D advection test problem with a monotonic Hermite interpolant. Using a tensor product cubic Lagrange interpolant in 2D requires 5 1D cubic interpolants, dependant on 16 points, and in 3D this increases to 21 cubic interpolants using 64 data points. For the ECMWF forecast model the interpolation for any points not directly surrounding the departure point is replaced by linear interpolation [40, §3a]. This is described as quasi-cubic interpolation. A 2D quasi-cubic interpolant uses 3 cubic and 2 linear interpolants, depending on 12 surrounding points, and a 3D quasi-cubic interpolant uses 7 cubic and 10 linear interpolants, depending on 32 surrounding points.

2.6 Viscous Burgers' Equation

In this thesis we will be considering problems with semi-Lagrangian schemes in the presence of sharp jumps. The viscous Burgers' equation with a small viscosity ε is a canonical nonlinear PDE with advection which, with the exception of monotonically increasing initial conditions, forms sharp fronts of width of order ε in finite time. This makes it an ideal model problem for us and we will focus much of our attention on it. Burgers' equation is used as a core example in LeVeque (1992) [29], and many of its properties are presented there, some of which we shall repeat.

The viscous Burgers' equation is given by

$$u_t + uu_x = \varepsilon u_{xx}\,, \tag{2.52}$$

$$u(x, t^0) = u_0(t)\,, \tag{2.53}$$

where ε is taken here to be a small positive "viscosity parameter", and subscripts t and x are taken to mean the partial derivatives with respect to t and x respectively.

It can be verified that

$$u(x,t) = c - \alpha \tanh\left(\frac{\alpha}{2\varepsilon}(x - ct)\right) \tag{2.54}$$

is a travelling wave solution to Burgers' equation with speed c and height parameter α. The travelling wave solution (2.54) satisfies the asymptotic boundary conditions

$$\begin{aligned} u(-\infty, t) &= c + \alpha\,, \\ u(\infty, t) &= c - \alpha\,. \end{aligned} \tag{2.55}$$

Solutions corresponding to other specific boundary and initial conditions can be found using the Cole-Hopf transformation [13], which transforms Burgers' equation into the linear heat equation.

Taking ε to zero in Burgers' equation leads to the inviscid Burgers' equation [29, (3.14)]

$$u_t + uu_x = 0\,. \tag{2.56}$$

For this equation the characteristics i.e. curves along which the solution is constant, are straight lines with gradient $u_0(x)$ for each x [29, (3.17)]. In other words, the solution is constant with value $u_0(s)$ along the characteristics which are given for each s by the

straight lines (x, t) with the equation

$$x = s + u_0(s)t\,. \tag{2.57}$$

When two characteristics intersect for the first time, a shock forms at that point and there is a discontinuity in the solution $u(x, t)$ [29, §3.3].

With a non-zero viscosity term, solutions to Burgers' equation (2.52) instead form *fronts* with a width of order ε: for example the travelling wave solution (2.54) jumps from $c + 0.95\alpha$ to $c - 0.95\alpha$ about the point

$$x^* = ct \tag{2.58}$$

over a distance of

$$\begin{aligned} w &= \frac{4\tanh^{-1}(0.95)\varepsilon}{\alpha} \\ &\approx \frac{7.3\varepsilon}{\alpha}\,, \end{aligned} \tag{2.59}$$

shown in Figure 2-8.

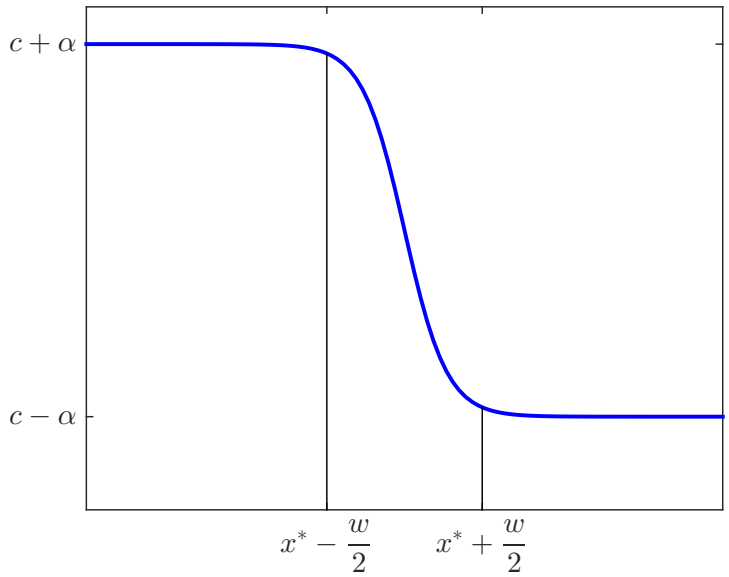

Figure 2-8: A travelling wave solution to the viscous Burgers' equation. The solution jumps from $c + 0.95\alpha$ to $c - 0.95\alpha$ over a front width w of order ε and travels in the positive x direction with speed c.

Reducing ε to zero in the travelling wave solution (2.54) gives a vanishing viscosity

solution to equation (2.56),

$$u(x,t) = \begin{cases} c+\alpha\,, & x < ct\,, \\ c-\alpha\,, & x > ct\,. \end{cases} \tag{2.60}$$

LeVeque [29, §3.5] shows how the solution in equation (2.60) is the unique weak solution to the Riemann Problem

$$\begin{aligned} u_t + uu_x &= 0\,, \\ u(x,0) &= \begin{cases} c+\alpha & x < 0 \\ c-\alpha & x > 0 \end{cases}, \end{aligned} \tag{2.61}$$

for $\alpha > 0$. When $\alpha < 0$, there are infinitely many weak solutions [29].

The speed of a shock can be determined from the Rankine-Hugoniot jump condition [29, (3.33)]: for a 1D conservation problem of the form

$$u_t + (f(u))_x = 0\,, \tag{2.62}$$

with a shock at x^* with left and right limits u_l and u_r respectively, the shock moves with speed c, where

$$f(u_l) - f(u_r) = c(u_l - u_r)\,. \tag{2.63}$$

We note that we can rewrite the inviscid Burgers' equation in the form of (2.62), as

$$u_t + \left(\frac{u^2}{2}\right)_x = 0\,, \tag{2.64}$$

so that $f(u) = \frac{u^2}{2}$. This implies that [29, (3.26)]

$$c = \frac{u_l + u_r}{2}\,. \tag{2.65}$$

This is in agreement with limit of the speed of the travelling wave solution in (2.60) with ε taken to 0.

The second half of LeVeque (1992) [29] is dedicated to numerical methods, such as conservative methods (§12.1), discrete conservation (§12.3) and Roe's approximate Riemann solver (§14.2), appropriate for inviscid Burgers' equation.

Here we shall be concentrating on using semi-Lagrangian methods to solve the viscous Burgers' equation.

Burgers' equation (2.52) can be expressed with a Lagrangian derivative (2.1) as

$$\frac{\mathrm{D}u}{\mathrm{D}t} = \varepsilon u_{xx}\,, \tag{2.66}$$

with kinematic equation

$$\begin{aligned}\frac{\mathrm{D}x}{\mathrm{D}t} &= \frac{\partial x}{\partial t} + u\frac{\partial x}{\partial x} \\ &= u(x,t)\,.\end{aligned} \tag{2.67}$$

This will be the subject of Chapter 3.

As stated earlier, Burgers' equation is a common test problem for advective schemes.

Kuo and Williamson (1990) [24] used a 3TL semi-Lagrangian scheme applied to the inviscid Burgers' equation. They used cubic spine interpolation, but reported seeing similar results when using the monotonicity preserving Hermite interpolants introduced by [22]. The initial conditions were chosen such that a singularity forms at $t = 1$, and show that leading up to this time, the errors are localised around the shock.

Bermejo and Staniforth (1992) [2, §3d] repeat these experiments, again with cubic splines, but also with their flux-limiter (as discussed above in Section 2.5.4). They track the solution past shock-formation using a shock-tracking algorithm, effectively imposing the Rankine-Hugoniot condition on the numerical solution. They show improvements on results obtained with the methods in [24].

Smith (2000) [47, §7.4] uses a 2TL SL scheme applied to the inviscid Burgers' equation up to front formation, and to the viscous Burgers' equation tracking a travelling wave solution using cubic Lagrange, quintic Lagrange and centred quadratic interpolants. The author observes a travelling wave but with a distorted front, showing the cubic interpolant to give the least error.

Chapter 3

Semi-Lagrangian Methods Applied to Burgers' Equation

3.1 Introduction

In this chapter we apply SL methods to Burgers' equation with a fixed spatial mesh and linear interpolation. We are primarily interested in analysing the performance of the method in this context. We achieve this by using a backwards error analysis of the modified equations which solve the discretised Burgers' equation. This correctly predicts a number of issues observed with SL methods. In particular our analysis accounts for the diffusive effect related to its interpolation error, and the change in wave speed when considering travelling wave solutions to Burgers' equation. Later in Chapter 5 we show how both these errors can be reduced by using a moving mesh method.

3.1.1 Structure of Chapter

This chapter is structured as follows: In Section 3.2 we remind ourselves of the 1D viscous Burgers' equation and travelling wave solutions, and the Semi-Lagrangian discretisation. In Section 3.3 we provide details of an implementation of the SL method for Burgers' equation and show a motivating example highlighting the different front-width $\widehat{\varepsilon}$ and front-speed $\widehat{c}$. In Section 3.4 we analyse the SL discretisation with linear interpolation, find the modified equation which the numerical solution satisfies to leading order, and provide equations for $\widehat{\varepsilon}$ and $\widehat{c}$. In Section 3.5 we show good agreement between the expressions for $\widehat{\varepsilon}$ and $\widehat{c}$, and numerical experiments for small fronts, and qualitative agreement when using cubic Lagrange interpolation. Finally in Section 3.6 we present our conclusions and discuss the implications of this work on moving meshes

for SISL NWP.

3.2 Background

Burgers' equation in 1D,

$$u_t + uu_x = \varepsilon u_{xx}\,, \tag{3.1}$$

where ε is a small positive parameter is a nonlinear PDE with advection which can develop steep fronts from smooth initial conditions, making it an ideal candidate for use as a test problem.

As described in equations (2.66) and (2.67) Burgers' equation can be expressed with Lagrangian derivatives from (2.1) as

$$\frac{\mathrm{D}u}{\mathrm{D}t} = \varepsilon u_{xx}\,, \tag{3.2}$$

$$\frac{\mathrm{D}x}{\mathrm{D}t} = u\,. \tag{3.3}$$

3.2.1 Travelling Wave (TW) Solution

A solution to Burgers' equation which is also a travelling wave is

$$u(x,t) = c - \alpha \tanh\left(\frac{\alpha}{2\varepsilon}(x-ct)\right)\,. \tag{3.4}$$

This travelling wave solution has a number of key properties which make it extremely useful as a test problem with an exact solution. An appropriate set of asymptotic boundary conditions which agree with this exact travelling wave solution are

$$\lim_{x\to\pm\infty}\left[u(x,t)\right] = c \mp \alpha\,. \tag{3.5}$$

The solution (3.4) varies from $c + 0.95\alpha$ to $c - 0.95\alpha$ over a *front-width* of

$$w = \frac{4\varepsilon}{\alpha}\tanh^{-1}(0.95)\,, \tag{3.6}$$

centred around $x = ct$, where the gradient is

$$u_x(ct,t) = \frac{-\alpha^2}{2\varepsilon}\,. \tag{3.7}$$

Away from the front at $x = ct$, the solution rapidly approaches the boundary values given by (3.5). Hence it is simple to implement the boundary conditions to machine precision on finite domains enclosing the front.

In addition to this, we can explicitly solve the trajectory equation (3.3) for the travelling wave solution, giving

$$\vec{x}(t) = \frac{2\varepsilon}{\alpha} \tanh^{-1}\left(\phi_0 \exp\left(\frac{-\alpha^2 t}{2\varepsilon}\right)\right) + ct\,, \tag{3.8}$$

where

$$\phi_0 = \tanh\left(\frac{\alpha\vec{x}(0)}{2\varepsilon}\right)\,. \tag{3.9}$$

For convenience, some derivatives of Burgers' equation and the travelling wave solution can be found in Appendix B.1.

3.3 Implementation and Results

The SL method, as described in Chapter 2, was applied to Burgers' equation in [47, §7.4]. It was noted that it gives errors in wave speed and shape, which depend on the choice of interpolant and the departure point calculation. We now implement a semi-Lagrange method for Burgers' equation and give an analytical theory for these errors when using linear interpolation. These errors are compared to the errors from numerical experiments, and we show that the errors are qualitatively similar for higher order interpolants.

We consider a SL discretisation of Burgers' equation, as described in detail in Chapter 2. The discretisation is a two-time-level finite difference scheme. Since Burgers' equation is linear in a Lagrangian frame of reference, there is no requirement for an inner loop. The two-time-level off-centred temporal discretisation of Burgers' equation (3.10) is

$$U_A^{n+1} - \theta_u \Delta t \varepsilon (U_{xx})_A^{n+1} = U_D^n + (1-\theta_u)\,\Delta t \varepsilon (U_{xx})_D^n\,, \tag{3.10}$$

and the kinematic equation (3.3) is discretised as

$$X_D = x_A - \Delta t\left[\theta_x U_A^{n+1} + (1-\theta_x)\,U_D^n\right]\,. \tag{3.11}$$

We have included the off-centring parameter θ_u, as in equation (2.15), and an independent off-centring parameter for the kinematic equation, θ_x, which was taken in equation (2.30) as $\theta_x = 1/2$.

We now show a motivating example, numerically approximating a travelling wave

solution to Burgers' equation. We consider

$$u_t + uu_x = \varepsilon u_{xx}\,, \quad (x,t) \in [-1,4] \times [0,1.5]\,, \tag{3.12}$$

with boundary and initial conditions

$$u(x,0) = c - \alpha \tanh\left(\frac{\alpha x}{2\varepsilon}\right)\,, \tag{3.13}$$

$$u(-1,t) = c - \alpha \tanh\left(\frac{\alpha(-1-ct)}{2\varepsilon}\right) \approx c + \alpha\,, \tag{3.14}$$

$$u(4,t) = c - \alpha \tanh\left(\frac{\alpha(4-ct)}{2\varepsilon}\right) \approx c - \alpha\,. \tag{3.15}$$

The boundary condition approximations in (3.14) and (3.15) are accurate to within machine precision for $t \in [0,3]$ provided $\alpha > 40\varepsilon$, hence in practice we use these.

We discretise $u(x,t)$ in time and space as

$$U_i^n \approx u(-1 + i\Delta x, n\Delta t) \tag{3.16}$$

with

$$\Delta x = \frac{5}{N_x + 1}\,, \tag{3.17}$$

$$\Delta t = \frac{1.5}{N_t}\,. \tag{3.18}$$

We use finite difference and linear interpolation to evaluate $\mathbf{U}^n$ and $(\mathbf{U}_{xx})^n$ at the departure point $\mathbf{X}_D$. Relative to a coarse mesh-width, the travelling wave solution has a low-smoothness point at $x = ct$. As stated in Section 2.5.4 this can lead to oscillations in higher-order interpolants, hence our analysis will be focused on linear interpolation, though we also show numerical results for other interpolants.

The 2TL semi-Lagrangian discretisation with finite difference in space is

$$\mathbf{U}^{n+1} - \theta_u \Delta t \varepsilon(\delta_x^2 \mathbf{U}^{n+1} + \mathbf{b}_{\delta_x^2}) = \left[\mathbf{U}^n + (1-\theta_u)\Delta t \varepsilon(\delta_x^2 \mathbf{U}^n + \mathbf{b}_{\delta_x^2})\right]_D\,, \tag{3.19}$$

where δ_x^2 is the tridiagonal second order finite difference matrix

$$\delta_x^2 = \frac{1}{\Delta x^2}\begin{pmatrix} -2 & 1 & & \\ 1 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ & & 1 & -2 \end{pmatrix}\,, \tag{3.20}$$

and $\mathbf{b}_{\delta_x^2}$ is the boundary condition correction

$$\mathbf{b}_{\delta_x^2} = \Delta x^{-2}(u(-1,t) \quad 0 \quad \dots \quad 0 \quad u(4,t))^T . \tag{3.21}$$

This discretisation (3.19) is augmented by the discrete kinematic equation

$$\mathbf{X}_D = \mathbf{x}_A - \Delta t \left(\theta_x \mathbf{U}^{n+1} + (1-\theta_x)\mathbf{U}_D^n\right) . \tag{3.22}$$

The solution procedure is given in Figure 3-1, and implemented in the code `burg2.m` in Appendix A.1.

```
1: for n := 1 to N_t do
2:    estimate X_D from U^n
3:    for k := 1 to K do
4:       Interpolate U^n and δ_x^2(U^n) onto X_D
5:       Solve the N_x-by-N_x tridiagonal system (3.19) to find an esti-
         mate to U^{n+1} for the given departure point
6:       Update estimate to X_D from the kinematic equation (3.22)
7:    end for
8: end for
```

Figure 3-1: Solution procedure for the Semi-Lagrangian Burgers' equation code.

The code was run with $\varepsilon = 10^{-4}$, $c = 1$, $\alpha = 0.1$, $N_x = 100$, $N_t = 40$ and $\theta_u = \theta_x = 0.5$. The solution at $t = 1.5$ with the exact travelling wave solution can be seen in Figure 3-2.

For the front speed, we look at the centre of the front, $x^*(t)$, which we define to be the point on the piecewise linear interpolant of $\mathbf{U}$ such that

$$U(x^*(t), t) = c\,, \tag{3.23}$$

and observe it has travelled further than the centre of the analytic front. We can obtain an estimate of the numerical speed $\widehat{c}$ by tracking $x^*(t)$ and calculating the gradient of the line of best fit. This results in a value of $\widehat{c} \approx 1.0102$. A finite difference approximation to the speed with the gradient of the line of best fit is seen in Figure 3-3.

For the numerical viscosity $\widehat{\varepsilon}$, we use the gradient of U at x^* and equation (3.7) to obtain

$$\widehat{\varepsilon} = \frac{-\alpha^2}{2U_x(x^*(t), t)}\,, \tag{3.24}$$

which gives us a value of $\widehat{\varepsilon} \approx 0.0052$, larger than the analytic parameter $\varepsilon = 10^{-4}$. We

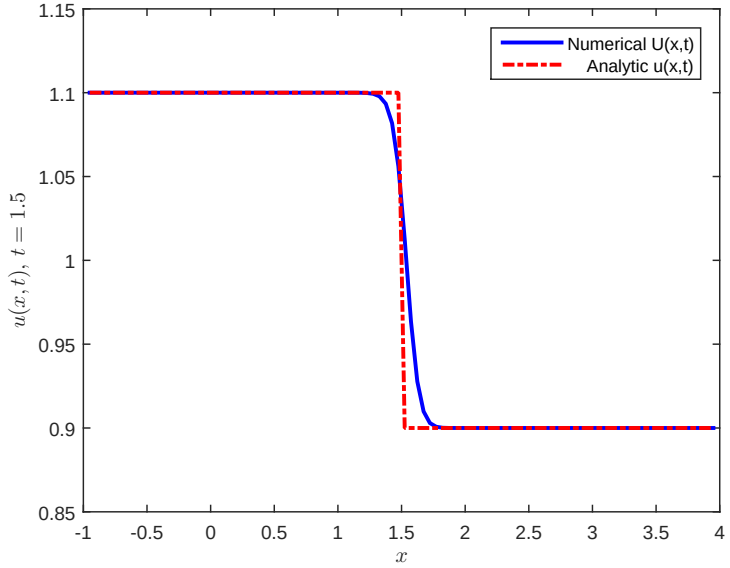

Figure 3-2: Comparison of the numerical and analytic fronts at time $t = 1.5$. We observe a tanh-like profile for the numerical solution, but with a slightly faster wave speed and a larger front-width.

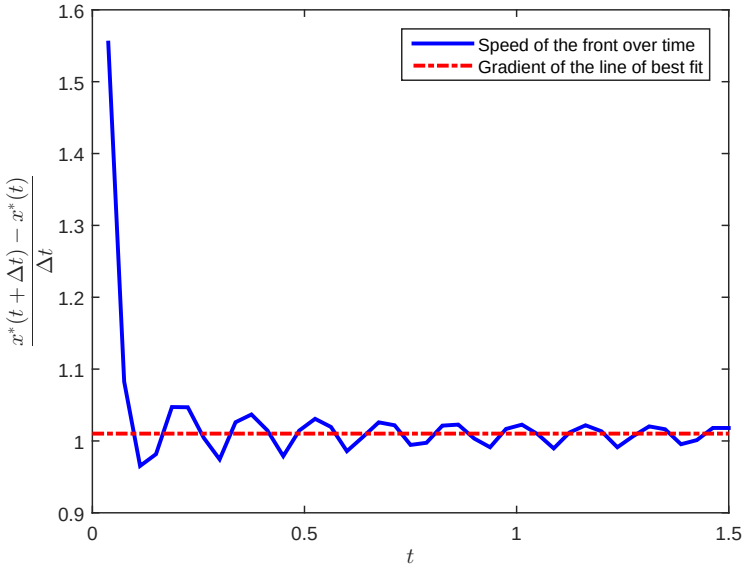

Figure 3-3: Comparison of the speed of the centre of the front $\dfrac{\mathrm{d}x^*}{\mathrm{d}t}$ and the estimate $\widehat{c} = 1.0102$. The trend is slightly faster than the speed of the exact solution $c = 1$.

could alternatively use the front-width $\widehat{w}$ and equation (3.6) to obtain

$$\widehat{\varepsilon} = \frac{\alpha \widehat{w}}{4 \tanh^{-1}(0.95)} , \tag{3.25}$$

which, in this case, gives $\widehat{\varepsilon} = 0.0048$.

3.4 Analysis

We assume that there is a travelling wave solution to the SL discretisation $v(x - \widehat{c}t)$, and that the numerical solution U is a discrete sampling of this solution. Our approach is to find, to leading order, the equation for v which satisfies the SL discretisation. We then analyse this equation, looking for the discrepancies observed in Figure 3-2.

Before this analysis, we first look at just the departure point error associated with the implementation of the kinematic equation.

3.4.1 Departure Point Error

Before looking at smooth solutions to the discretised equation we look at the error from the departure point discretisation on its own. The discretisation of the kinematic equation (3.11) with $\theta_x = 1/2$ is

$$x_A - X_D = \frac{\Delta t}{2} \left[u_A^{n+1} + u_D^n \right] . \tag{3.26}$$

We can look at the error after a single time step by substituting in the exact travelling wave solution (3.4) and comparing with the exact solution for $\vec{x}(t)$ from equation (3.8),

$$E_{\mathrm{D}}(x_A) = x_A - X_D^0 - \frac{\Delta t}{2} \left[u(x_A, \Delta t) + u(x_D, 0) \right] . \tag{3.27}$$

For $\varepsilon = 0.002$, $\Delta t = 0.01$, $c = 3$ and $\alpha = 1$, we see the absolute error in departure points in Figure 3-4. This is simply a demonstration of the availability of E_{D}; we do not analyse this error by itself, as it will be included in the full SL analysis later in this section.

We now analyse the full SL discretisation of both Burgers' equation and its associated kinematic equation.

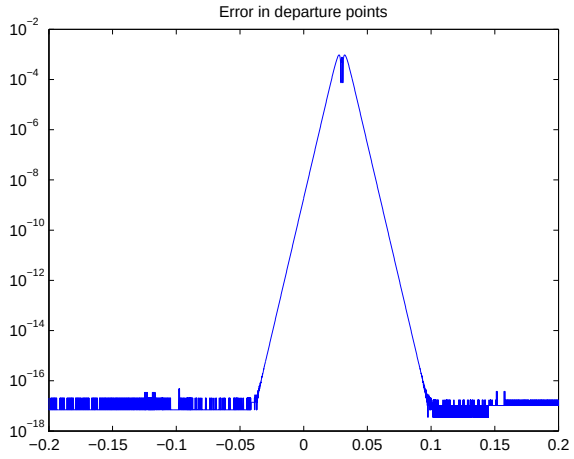

Figure 3-4: Departure point error, E_{D}.

3.4.2 Solution of the Modified Equation

We assume that the numerical solution $U(x,t)$ is a travelling wave, such that

$$U(x,t) = v(z) \tag{3.28}$$

with

$$z = x - \widehat{c}t\,. \tag{3.29}$$

where the profile $v(z)$ and the wave speed $\widehat{c}$ are to be found.

The terms Δx and Δt are assumed to be of the same order. Our analysis accounts for the departure point error and interpolation error up to order $\mathcal{O}((\Delta x + \Delta t)^2)$. The finite difference discretisation error from the viscosity term $\Delta t \varepsilon U_{xx}$ is, to leading order on the uniform grid $-(\varepsilon/12)\Delta t \Delta x^2 U_{xxxx}$. We assume that U_{xxxx} is bounded as $\Delta x, \Delta t \to 0$, hence we do not consider it in our analysis.

Our approach involves taking Taylor expansions of $v(z)$ about (x_A, t^n), substituting these terms into (3.10) and (3.11), then analysing the resulting equation for v, a *modified equation* (see [29, §11.1]).

Taylor Expansion at (x_A, t^{n+1})

Let x_j be a grid-point coinciding with an arrival point x_A. Now

$$U_A^n = v(x_j - \widehat{c}t^n) = v(z_j^n) \quad \text{where} \quad z_j^n := x_j - \widehat{c}t^n\,, \tag{3.30}$$

and for a travelling wave with speed $\widehat{c}$,

$$\begin{aligned} U_A^{n+1} &= v(z_j^n - \widehat{c}\Delta t) \\ &= v - \widehat{c}\Delta t v' + \frac{(\widehat{c}\Delta t)^2}{2} v'' + \mathcal{O}(\Delta t^3)\,. \end{aligned} \tag{3.31}$$

Here v is assumed to be evaluated at $z = z_j^n$, and v' represents derivative of v with respect to its argument, again evaluated at $z = z_j^n$. Similarly to (3.31),

$$(U_{xx})_A^{n+1} = v'' - \widehat{c}\Delta t v''' + \mathcal{O}(\Delta t^2)\,, \tag{3.32}$$

and we can express the left-hand side of (3.10) as

$$\begin{aligned} U_A^{n+1} - \Delta t \theta_u \varepsilon (U_{xx})_A^{n+1} = v + \Delta t \left(-\widehat{c} v' - \varepsilon \theta_u v''\right) \\ + \frac{\Delta t^2}{2}\left(\widehat{c}^2 v'' + 2\widehat{c}\varepsilon\theta_u v'''\right) + \mathcal{O}(\Delta t^3)\,. \end{aligned} \tag{3.33}$$

Taylor Expansion at (X_D, t^n)

For simplicity we assume that the solution is uniformly non-negative and that we have a uniform mesh. For a trajectory from X_D to x_A, we consider a non-dimensional advection Courant number

$$\nu_{\mathrm{CFL}} = \frac{u\Delta t}{\Delta x}\,. \tag{3.34}$$

Note that, as stated above, Δx and Δt are of a similar order, but ν_{CFL} will often be larger that 1. To first order, the kinematic equation (3.3) gives

$$\begin{aligned} x_A - X_D &= U\Delta t \\ &= \nu_{\mathrm{CFL}}\Delta x\,. \end{aligned} \tag{3.35}$$

We split up the advection relative to the mesh into an integer part $\mathcal{N} \in \mathbb{Z}$, and fractional part $y \in [0, 1)$, so that

$$x_A - X_D = (\mathcal{N} + y)\Delta x\,. \tag{3.36}$$

It is also useful to observe that $\mathcal{N} = \lfloor \nu_{\mathrm{CFL}} \rfloor$, where $\lfloor\cdot\rfloor$ represents the floor function, and that departure point X_D lies in the interval $(x_{j-\mathcal{N}-1}, x_{j-\mathcal{N}}]$. Two examples are given in Figure 3-5, which shows a sketch of two potential advection trajectories corresponding to $\mathcal{N} = 2$ and $\mathcal{N} = 0$. Using linear interpolation to evaluate $u(x_D, t^n)$ in the right-hand side of (3.10), we get

$$U_D^n = (1 - y)U(x_{j-\mathcal{N}}, t^n) + yU(x_{j-\mathcal{N}-1}, t^n)\,. \tag{3.37}$$

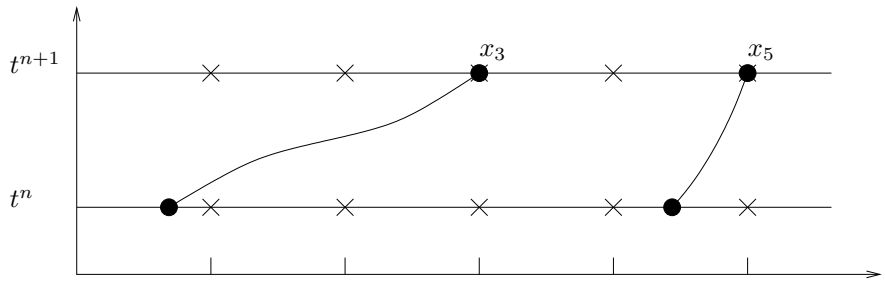

Figure 3-5: Example of advection Courant numbers. For the trajectory arriving at x_3, we have $\mathcal{N} = 2$, whilst the trajectory arriving at x_5 has $\mathcal{N} = 0$.

In terms of v, we have

$$U(x_{j-\mathcal{N}}, t^n) = v(z_j^n - \mathcal{N}\Delta x)\,, \tag{3.38}$$

$$U(x_{j-\mathcal{N}-1}, t^n) = v(z_j^n - (\mathcal{N}+1)\Delta x)\,, \tag{3.39}$$

and taking Taylor expansions about $z = z_j^n$ and substituting into (3.37) gives

$$U_D^n = v - (\mathcal{N}+y)\Delta x v' + \left((\mathcal{N}+y)^2 + (1-y)y\right)\frac{\Delta x^2}{2}v'' + \mathcal{O}(\Delta x^3)\,. \tag{3.40}$$

We now want to replace the Courant number $(\mathcal{N}+y)\Delta x$ with an expression involving v. Rewriting the departure point equation (3.11) as

$$x_A - X_D = \Delta t\left[\theta_x u_A^{n+1} + (1-\theta_x)\,u_D^n\right]\,, \tag{3.41}$$

and using the expressions for u_A^{n+1} and u_D^n from (3.31) and (3.40), (3.36) gives

$$\begin{aligned}(\mathcal{N}+y)\Delta x = \Delta t\Bigg(&\theta_x\left(v - \widehat{c}\Delta t v' + \frac{\widehat{c}^2\Delta t^2}{2}v''\right)\\ &+ (1-\theta_x)\left(v - (\mathcal{N}+y)\Delta x v' + \frac{(\mathcal{N}+y)^2\Delta x^2}{2}v'' + \frac{(1-y)y\Delta x^2}{2}v''\right)\Bigg)\\ &+ \mathcal{O}((\Delta x + \Delta t)^4)\,.\end{aligned} \tag{3.42}$$

We can truncate this further and substitute in a first order approximation

$$(\mathcal{N}+y)\Delta x = \Delta t v + \mathcal{O}(\Delta t^2)\,, \tag{3.43}$$

to get

$$(\mathcal{N}+y)\Delta x = \Delta t\left(v - (\theta_x v + (1-\theta_x)\,\widehat{c})\,\Delta t v'\right) + \mathcal{O}((\Delta x + \Delta t)^3)\,, \tag{3.44}$$

and

$$U_D^n = v - \Delta t v v' + \frac{\Delta t^2}{2}\left(2(\theta_x \widehat{c} + (1-\theta_x)v)v'^2 + v^2 v''\right) + \frac{\Delta x^2}{2}(1-y)yv'' + \mathcal{O}((\Delta x + \Delta t)^3)\,. \quad (3.45)$$

We take a moment to remark that the last Δx^2 term is equal to the leading order error for linear interpolation from equation (2.35).

Using a similar process, we get

$$(U_{xx})_D^n = v'' - \Delta t v v''' + \mathcal{O}((\Delta x + \Delta t)^2)\,. \quad (3.46)$$

Accordingly we express the right-hand side of (3.10) as

$$U_D^n + (1-\theta_u)\,\Delta t \varepsilon (U_{xx})_D^n = v - \Delta t\left(vv' - (1-\theta_u)\,\varepsilon v''\right) + \frac{\Delta t^2}{2}\left(2\left(\theta_x \widehat{c} + (1-\theta_x)\,v\right)v'^2 + v^2 v'' - 2\,(1-\theta_u)\,\varepsilon v v'''\right) + \frac{\Delta x^2}{2}(1-y)yv'' + \mathcal{O}((\Delta t + \Delta x)^3)\,. \quad (3.47)$$

Forming the Modified Equation

With the left- and right-hand side equations, (3.33) and (3.47), we can express the SISL discretisation (3.10) in terms of v, as

$$\Delta t\left((v-\widehat{c})v' - \varepsilon v''\right) = \frac{\Delta t^2}{2}\left((\widehat{c}+v)v'^2 + (v^2 - \widehat{c}^2)v'' - (\widehat{c}+v)\varepsilon v'''\right) + \frac{\Delta t^2}{2}(\widehat{c}-v)\left((2\theta_x - 1)\,v'^2 - (2\theta_u - 1)\,\varepsilon v'''\right) + \frac{\Delta x^2}{2}(1-y)yv'' + \mathcal{O}((\Delta t + \Delta x)^3)\,, \quad (3.48)$$

with boundary conditions

$$v(-\infty) = c + \alpha\,, \quad (3.49)$$

$$v(\infty) = c - \alpha\,. \quad (3.50)$$

Equation (3.48) is our full modified equation for a numerical travelling wave solution to the semi-Lagrangian discretisation of Burgers' equation with linear interpolation.

Using the first order approximation (3.43), we can instead express the interpolation term (the Δx^2 term) in (3.48) as

$$\frac{\Delta x^2}{2}(1-y)yv'' = \left(((2\mathcal{N}+1)\Delta x - v\Delta t)\, v\Delta t - (\mathcal{N}^2+\mathcal{N})\Delta x^2\right)\frac{v''}{2}\,. \tag{3.51}$$

We have now achieved our desired objective of obtaining the modified equation satisfied by v. The next step is to analyse equation (3.48) to get estimates of the numerical viscosity parameter and the numerical wave speed.

3.4.3 Asymptotic expansion of the modified equation

To analyse the modified equation, we shall perform a rescaling, followed by an expansion argument and isolate the modified viscosity $\widehat{\varepsilon}$ and modified wave speed $\widehat{c}$. We simplify our analysis by only considering the case of a Crank-Nicholson-like discretisation with $\theta_x = \theta_u = \frac{1}{2}$.

The full modified equation is

$$\begin{aligned}\Delta t\left((v-\widehat{c})\,v' - \varepsilon v''\right) = \frac{\Delta t^2}{2}\left(\widehat{c}+v\right)\left(v'^2 + (v-\widehat{c})\,v'' - \varepsilon v'''\right)\\ + \frac{1}{2}\left[((2\mathcal{N}+1)\,\Delta x - v\Delta t)\, v\Delta t - \left(\mathcal{N}^2+\mathcal{N}\right)\Delta x^2\right]v''\\ + \mathcal{O}\left((\Delta x+\Delta t)^3\right)\,.\end{aligned} \tag{3.52}$$

Motivated by the exact solution (3.4)

$$u(x,t) = c - \alpha\tanh\left(\frac{\alpha(x-ct)}{2\varepsilon}\right)\,, \tag{3.53}$$

we introduce the scaling

$$v(z) = c + \alpha W(s)\,, \tag{3.54}$$

where

$$s = \alpha z\,. \tag{3.55}$$

We now consider the leading order modified equation, ignoring the $\mathcal{O}((\Delta x+\Delta t)^3)$ terms. The modified equation (3.52) can be expressed in the rescaled variables $W(s)$

and takes the form

$$\begin{aligned}\Delta t\left(\alpha^2\left(c-\widehat{c}+\alpha W\right)W'-\alpha^3\varepsilon W''\right)=&\\ \frac{\Delta t^2}{2}\left(c+\widehat{c}+\alpha W\right)\left(\alpha^4 W'^2+\alpha^3\left(c-\widehat{c}+\alpha W\right)W''-\alpha^4\varepsilon W'''\right)&\\ +\frac{\alpha^3\Delta t}{2}\left[\left((2\mathcal{N}+1)\,\Delta x-(c+\alpha W)\Delta t\right)(c+\alpha W)-\left(\mathcal{N}^2+\mathcal{N}\right)\frac{\Delta x^2}{\Delta t}\right]W''\,,&\quad(3.56)\end{aligned}$$

where a prime now represents a derivative with respect to the rescaled variable s. This gives an equation for the modified equation in terms of $\widehat{c}$ and $W(s)$ with varying powers of α.

We investigate this rescaled modified equation by formally expanding $\widehat{c}$ and $W(s)$ in terms of α:

$$\widehat{c}=\widehat{c}_0+\alpha\widehat{c}_1+\alpha^2\widehat{c}_2+\cdots\,, \tag{3.57}$$

$$W(s)=W_0(s)+\alpha W_1(s)+\alpha^2 W_2(s)+\cdots\,. \tag{3.58}$$

The boundary conditions of the modified equation

$$\lim_{z\to\mp\infty}v(z)=c\pm\alpha\,, \tag{3.59}$$

give boundary conditions for the terms of the expansion as

$$\begin{aligned}\lim_{s\to\mp\infty}W_0(s)&=\pm 1\,,\\ \lim_{s\to\mp\infty}W_i(s)&=0\,,\quad\text{for } i>0\,.\end{aligned} \tag{3.60}$$

In searching for a heteroclinic solution, as with the exact solution, we also observe that all derivatives of $W_i(s)$ tend to zero as s tends to $\pm\infty$ for all $i=0,1,\ldots$.

Substituting the expansions (3.57) and (3.58) into equation (3.56), we can consider terms in successive powers of α.

Terms of order $\mathcal{O}(\alpha^2)$

With the above expansions for $W(s)$ and $\widehat{c}$ in the rescaled modified equation (3.56), considering only terms of order $\mathcal{O}(\alpha^2)$, we have

$$\alpha^2\Delta t(c-\widehat{c}_0)W_0'(s)=0\,. \tag{3.61}$$

For non-trivial solutions in $W_0(s)$, this leads to

$$\widehat{c}_0 = c\,. \tag{3.62}$$

Terms of order $\mathcal{O}(\alpha^3)$

Considering terms of order $\mathcal{O}(\alpha^3)$, we see that

$$\alpha^3 \Delta t \left((W_0 - \widehat{c}_1) W_0' - \tilde{\varepsilon} W_0'' \right) = 0\,, \tag{3.63}$$

where

$$\tilde{\varepsilon} := \varepsilon + \frac{1}{2} \left(((2\mathcal{N}+1)\Delta x - c\Delta t)\, c - (\mathcal{N}^2 + \mathcal{N}) \frac{\Delta x^2}{\Delta t} \right) . \tag{3.64}$$

Integrating equation (3.63) over $[-\infty, \infty]$ gives

$$\left[\frac{W_0(s)^2}{2} - \widehat{c}_1 W_0(s) - \tilde{\varepsilon} W_0'(s) \right]_{-\infty}^{\infty} = 0\,. \tag{3.65}$$

Substituting in the boundary conditions (3.60) gives

$$\widehat{c}_1 = 0\,. \tag{3.66}$$

The equation (3.63) together with the boundary conditions (3.60) has the well known heteroclinic solution

$$W_0(s) = -\tanh\left(\frac{s}{2\tilde{\varepsilon}} \right) . \tag{3.67}$$

This gives the numerical solution $v(z)$, to leading order of α

$$v(z) = c - \alpha \tanh\left(\frac{\alpha z}{2\tilde{\varepsilon}} \right) + \mathcal{O}(\alpha^2)\,, \tag{3.68}$$

suggesting a tanh-profile similar to the exact solution, but with a front-width parameter $\tilde{\varepsilon}$. This is in agreement with the results observed in Section 3.3.

Terms of order $\mathcal{O}(\alpha^4)$

We consider the next next order of the modified equation (3.56), $\mathcal{O}(\alpha^4)$:

$$\begin{aligned} \alpha^4 \Delta t \left((W_1 - \widehat{c}_2) W_0' + W_0 W_1' - \tilde{\varepsilon} W_1'' \right) = \\ \alpha^4 \Delta t^2 c \left((W_0')^2 + W_0 W_0'' - \varepsilon W_0''' \right) \\ + \alpha^4 \frac{\Delta t}{2} \left((2\mathcal{N}+1)\Delta x - 2c\Delta t \right) W_0 W_0''\,. \end{aligned} \tag{3.69}$$

Integrating this equation over $[-\infty, \infty]$ leads to the expression

$$\begin{aligned}\left[-\widehat{c}_2 W_0(s) + W_1(s)W_0(s) - \tilde{\varepsilon} W_1'(s)\right]_{-\infty}^{\infty} = \\ \Delta t c \left[W_0(s)W_0'(s) - \varepsilon W_0''(s)\right]_{-\infty}^{\infty} \\ + \frac{1}{2}\left((2\mathcal{N}+1)\Delta x - 2c\Delta t\right)\int_{-\infty}^{\infty} W_0(s)W_0''(s)\mathrm{d}s\,. \quad (3.70)\end{aligned}$$

We can explicitly evaluate the integral using the solution from (3.67). Using this result and rearranging gives

$$\widehat{c}_2 = \frac{-1}{6\tilde{\varepsilon}}((2\mathcal{N}+1)\Delta x - 2c\Delta t)\,, \quad (3.71)$$

Hence, we can reconstruct the expansion for $\widehat{c}$ (3.57) from (3.62), (3.66) and (3.71), giving

$$\widehat{c} = c - \frac{\alpha^2}{6\tilde{\varepsilon}}\left((2\mathcal{N}+1)\Delta x - 2c\Delta t\right) + \mathcal{O}(\alpha^3)\,. \quad (3.72)$$

Considering again the $\mathcal{O}(\alpha^4)$ equation, with a bit of calculus we can explicitly find an expression for $W_1(s)$. Equation (3.69) can be rearranged as

$$(W_0W_1)' - \tilde{\varepsilon}W_1'' = \Delta t c(W_0W_0' - \varepsilon W_0'')' + \widehat{c}_2(W_0' - 3\tilde{\varepsilon}W_0W_0'')\,. \quad (3.73)$$

With the observations that

$$W_0' = \frac{1}{2\tilde{\varepsilon}}(W_0^2 - 1) \quad (3.74)$$

and

$$W_0'' = \frac{1}{\tilde{\varepsilon}}W_0W_0'\,, \quad (3.75)$$

after integrating equation (3.73) and some careful manipulation, we get

$$W_1' - \frac{W_0}{\tilde{\varepsilon}}W_1 = \frac{1}{\tilde{\varepsilon}}\left(2\widehat{c}_2 - \Delta t c\frac{\tilde{\varepsilon}-\varepsilon}{\tilde{\varepsilon}^2}\right)W_0W_0' + \text{constant}\,. \quad (3.76)$$

If we consider s going to ∞ then we have the constant is 0. Using the integrating factor

$$\begin{aligned}\mathrm{I} &= \exp\left(-\int\frac{W_0(s)}{\tilde{\varepsilon}}\,\mathrm{d}s\right) \\ &= \frac{-1}{2\tilde{\varepsilon}W_0'}\,,\end{aligned} \quad (3.77)$$

we can integrate equation (3.76) to get

$$\frac{W_1}{W_0'} = \left(2\widehat{c}_2 - \Delta t c \frac{\tilde{\varepsilon} - \varepsilon}{\tilde{\varepsilon}^2}\right)\left(\ln\left(\cosh\left(\frac{s}{2\tilde{\varepsilon}}\right)\right) + d\right) . \tag{3.78}$$

If $W_1(s)$ makes no contribution to the error at $s = 0$, then $d = 0$. Using the equations for $W_0(s)$ and $\widehat{c}_2$, (3.67) and (3.71), we arrive at

$$W_1(s) = \frac{-1}{3\tilde{\varepsilon}}\left((2\mathcal{N}+1)\Delta x + \Delta t c \frac{\tilde{\varepsilon} - 3\varepsilon}{\tilde{\varepsilon}}\right)\ln\left(\cosh\left(\frac{s}{2\tilde{\varepsilon}}\right)\right)\operatorname{sech}^2\left(\frac{s}{2\tilde{\varepsilon}}\right) . \tag{3.79}$$

3.4.4 Further correction to ε

Before summarising our analysis, we consider a further restriction on the numerical viscosity parameter due to the mesh spacing. Assuming a tanh profile, we have a relationship from equation (3.24) between m, the gradient of the numerical solution and $\widehat{\varepsilon}$ the numerical viscosity term, namely

$$\widehat{\varepsilon} = \frac{-\alpha^2}{2m} . \tag{3.80}$$

When attempting to represent a tanh profile on a mesh with $\varepsilon \ll \Delta x$, provided we create no new extrema, the front almost surely jumps from $c + \alpha$ to $c - \alpha$ in a single interval, since the width of the moving front is small compared to the mesh width. This gives a maximum gradient $-2\alpha/\Delta x$. As such, this gradient corresponds to a minimum distinguishable viscosity parameter which we can represent with mesh spacing Δx,

$$\varepsilon_{\min} = \frac{\alpha \Delta x}{4} . \tag{3.81}$$

Taken with the above estimate for $\tilde{\varepsilon}$ in equation (3.64), we predict the numerical viscosity to be

$$\widehat{\varepsilon} = \max\left(\varepsilon + ((2\mathcal{N}+1)\Delta x - v\Delta t)\frac{v}{2} - (\mathcal{N}^2 + \mathcal{N})\frac{\Delta x^2}{2\Delta t}, \frac{\alpha \Delta x}{4}\right) . \tag{3.82}$$

We refer to this restriction on the minimum of $\widehat{\varepsilon}$ as an *aliasing problem.*

3.4.5 Summary of Analysis

Using Taylor series we produced a modified equation for travelling wave solution of the SISL discretisation of Burgers' equation with linear interpolation. After a rescaling motivated by the exact travelling wave solution to Burgers' equation, we considered expansions in terms of α. By considering terms of orders of α we arrived at expressions

for $v(z)$, $\tilde{\varepsilon}$ and $\widehat{c}$. Finally, referring to the minimum measurable viscosity parameter on a given spatial mesh, we arrive at expressions for $v(z)$, $\widehat{\varepsilon}$ and $\widehat{c}$ as

$$v(z) = c - \alpha \tanh\left(\frac{\alpha z}{2\widehat{\varepsilon}}\right) + \mathcal{O}\left(\alpha^2\right) , \tag{3.83}$$

$$\widehat{\varepsilon} = \max\left(\varepsilon + ((2\mathcal{N}+1)\Delta x - v\Delta t)\,\frac{v}{2} - (\mathcal{N}^2+\mathcal{N})\frac{\Delta x^2}{2\Delta t}\,,\, \frac{\alpha\Delta x}{4}\right) , \tag{3.84}$$

$$\widehat{c} = c - \frac{\alpha^2}{6\widehat{\varepsilon}}\left((2\mathcal{N}+1)\Delta x - 2c\Delta t\right) + \mathcal{O}\left(\alpha^3\right) . \tag{3.85}$$

To compare this with our numerical experiments we consider the model parameters as used in the motivational examples in Section 3.3, namely $c = 1$, $\alpha = 0.1$, $\varepsilon = 10^{-4}$, $\Delta x = \frac{5}{N_x+1}$ and $\Delta t = \frac{1.5}{N_t}$. In Figure 3-6 we see the theoretical numerical viscosity parameter $\widehat{\varepsilon}$ for different values of N_x and N_t. In each case $u(x,t)$ is assumed to be sufficiently close to c, (i.e. α is small) such that in the equation (3.82) we have

$$v \approx c\,, \tag{3.86}$$

$$\mathcal{N} \approx \left\lfloor \frac{c\Delta t}{\Delta x} \right\rfloor . \tag{3.87}$$

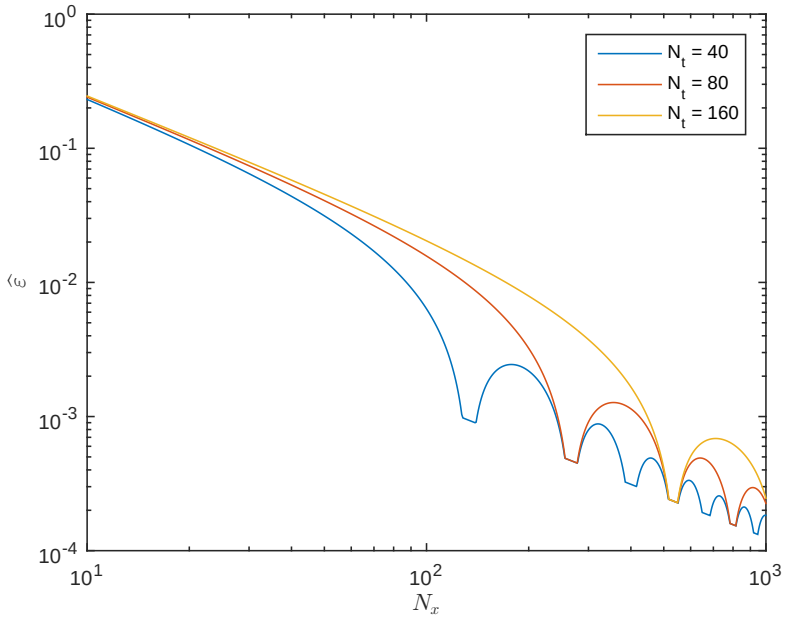

Figure 3-6: Theoretical approximation to $\widehat{\varepsilon}$ from equation (3.82), with $c = 1$, $\alpha = 0.1$ and $\varepsilon = 10^{-4}$. The "fingers" correspond to ν_{CFL} passing through integer values.

Using the same parameters as Figure 3-6, $c = 1$, $\alpha = 0.1$, $\varepsilon = 10^{-4}$, $\Delta x = \frac{5}{N_x+1}$ and $\Delta t = \frac{1.5}{N_t}$, and taking $\mathcal{N} \approx \left\lfloor \frac{c\Delta t}{\Delta x} \right\rfloor$ we can plot the theoretical wave speed $\widehat{c}$ for different N_x and N_t, which we see in Figure 3-7.

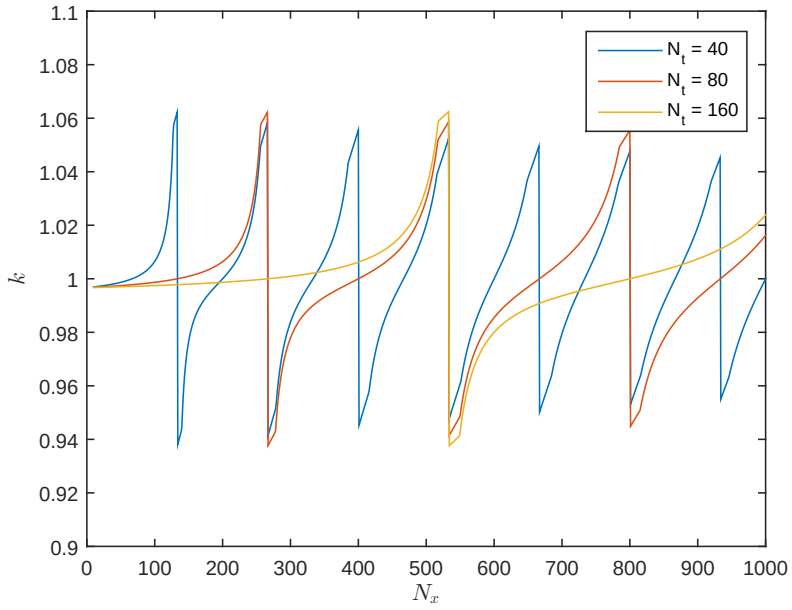

Figure 3-7: Theoretical estimate of $\widehat{c}$ from equation (3.72) for $c = 1$, $\alpha = 0.1$ and $\varepsilon = 10^{-4}$. We see a periodic behaviour as ν_{CFL} passes through integer values, although these peaks are getting smaller as N_x is increasing (next figure).

This error decays slowly with Δx and Δt. This can be seen for Δx by repeating the plot for $\widehat{c}$ from Figure 3-10, but with N_x taken larger, and plotted with a logarithmic scale in N_x, as seen in Figure 3-8.

3.5 Numerical Comparisons

We now run numerical simulations of Burgers' equation with a SL discretisation for different numbers of spatial and temporal points, with N_x from 10 to 1000, and N_t as 40, 80 and 160. In all cases we use viscosity parameter $\varepsilon = 10^{-4}$, domain $(x, t) \in [-1, 4] \times [0, 1.5]$, initial conditions $u(x, 0) = c - \alpha \tanh(\alpha x/(2\varepsilon))$ and boundary conditions $u(-1, t) = c + \alpha$, $u(4, t) = c - \alpha$. We run these experiments with a number of configurations, starting with the parameters and settings used in the motivating examples and analysis.

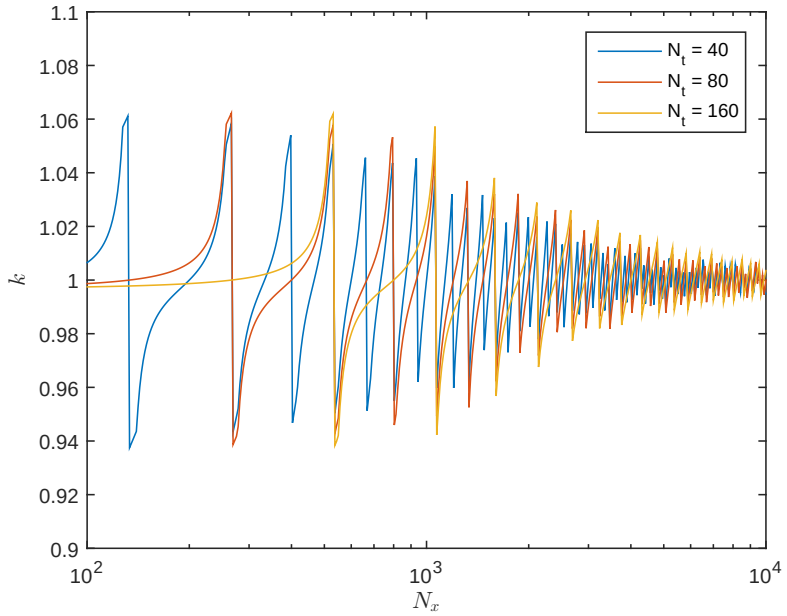

Figure 3-8: Theoretical estimate of $\widehat{c}$, as in Figure 3-7, but for more values of N_x and a logarithmic x–axis. The tend of the peaks decreasing is now apparent.

We run the experiments with $c = 1$, $\alpha = 0.1$, $\theta_x = \theta_u = \frac{1}{2}$, $\Delta x = \frac{5}{N_x+1}$ and $\Delta t = \frac{1.5}{N_t}$, and use piecewise linear interpolation.

The numerical viscosity is presented in Figure 3-9. We observe very good agreement with the theoretical estimate of $\widehat{\varepsilon}$ seen in Figure 3-6, including the periodicity with respect to N_x as the Courant number increases through integer values, the symmetry for an identical Courant number, and the minimum viscosity $\varepsilon_{\min}$ from equation (3.81). The two graphs from Figures 3-9 and 3-6 are plotted on the same axes in Figure 3-11, and we observe smaller error than predicted for small N_x, and slightly larger errors than the minimum aliasing errors.

From the same numerical experiments, we show the numerical wave speed in Figure 3-10. Again, qualitatively we see good agreement with the theoretical estimate, see in Figure 3-7. For $N_t = 40$ and $N_x < 200$ we have a larger wave-speed error than predicted. For larger N_x we see smaller peaks in the wave speed error, and less sharp jumps from faster-than to slower-than c than the theoretical estimate. These differences from the theory could be due to the weakening of the assumption that $u(x,t) \approx c$ and $\mathcal{N}$ is constant,

$$\mathcal{N} = \left\lfloor \frac{c\Delta t}{\Delta x} \right\rfloor \qquad \forall x, t\,. \tag{3.88}$$

The theory assumes that all departure points have the same $\mathcal{N}$ for a given N_x. In

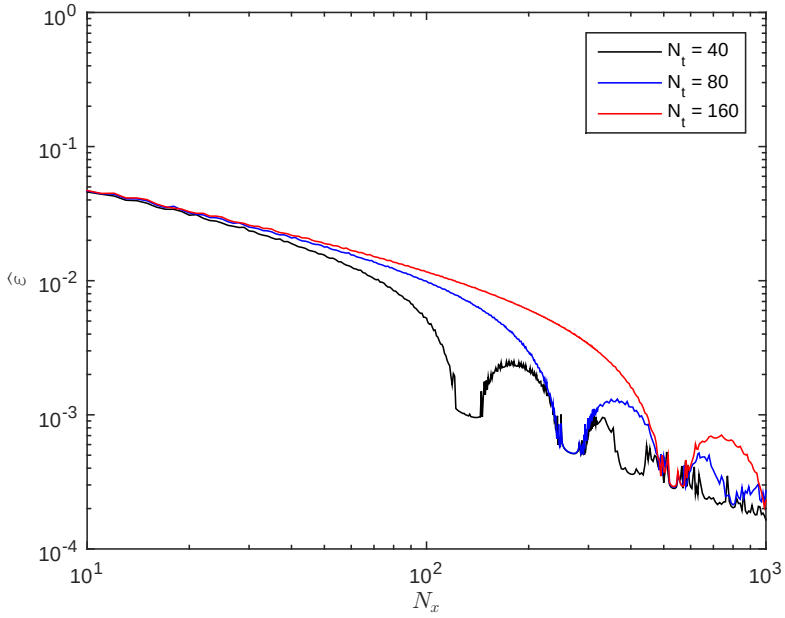

Figure 3-9: Numerically calculated viscosity parameter $\widehat{\varepsilon}$ for $\alpha = 0.1$, $\theta_x = \theta_u = \frac{1}{2}$ using linear interpolation. Except for small N_x this is in good agreement with the theoretical results in Figure 3-6 (plotted together in Figure 3-11).

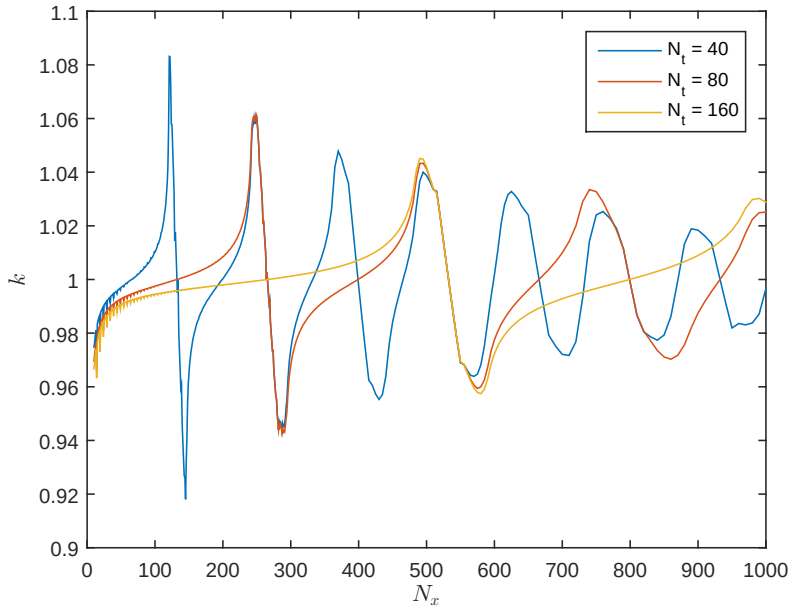

Figure 3-10: Numerically calculated $\widehat{c}$ for $\alpha = 0.1$, $\theta_x = \theta_u = \frac{1}{2}$ using linear interpolation. We observe similar behaviour to the theoretical wave speed, but the peaks are smeared over more N_x and decay with N_x faster than predicted.

practice, when ν_{CFL} is close to an integer value (fixed ν_{CFL}), some departure points will lead to the linear interpolation underestimating the true value, whilst others will lead to an overestimate. This would explain the increased decay of the maximum deviation of the speed from c, and the apparent *smearing* over N_x of the numerically calculated speed. Again, the theoretical and numerical wave speeds are plotted on the same axes in Figure 3-11.

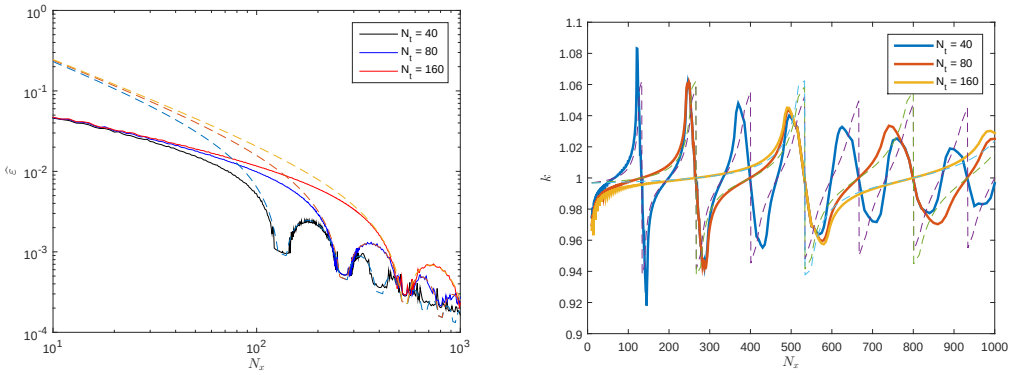

Figure 3-11: Numerically calculated and theoretical viscosity parameter $\widehat{\varepsilon}$ (left) and speed $\widehat{c}$ (right) for $\alpha = 0.1$, $\theta_x = \theta_u = \frac{1}{2}$, with linear interpolation. In both cases the numerical results are solid and the theoretical results are dashed. See Figures 3-6, 3-9, 3-7 and 3-10 for detail.

Plots for $\widehat{\varepsilon}$ and $\widehat{c}$ with $\theta_u = \theta_x = 0.55$ are practically indistinguishable (under visual inspection) from those in Figures 3-9 and 3-10.

Next, we repeated the experiments using cubic-Lagrange interpolation, described in Section 2.5, both with and without a flux limiter, described in Section 2.5.4. The results are in Figures 3-12, 3-13, 3-14 and 3-15.

Each of these graphs shows good qualitative agreement with those for the linear interpolant. The error in wave speed is slightly worse, while the numerical viscosity has better convergence with respect to Δx.

In Figures 3-16 and 3-17 we see the numerical viscosity and wave speed for $\alpha = 0.25$ and $\alpha = 0.5$, with linear interpolation. The aliasing error becomes the dominant term in the numerical viscosity term, whilst there is considerable slowing in the wave speed for Courant number $\nu_{\text{CFL}} > 1$. In these cases, the speed U varies in the domain from 1.5 to 0.5, and hence the value of $\mathcal{N}$ changes by a factor of 3, further weakening the assumption that $\mathcal{N}$ is approximately constant throughout the domain.

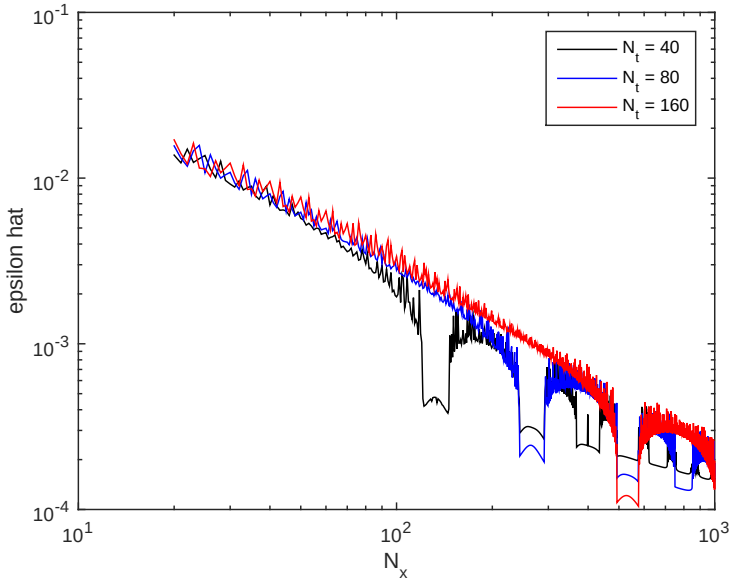

Figure 3-12: Calculated $\widehat{\varepsilon}$, as in Figure 3-9, but with cubic Lagrange interpolation. The aliasing minimum as described in Subsection 3.4.4 is reduced when no flux limiter is used, since the numerical solution increases above $c + \alpha$ to the left of the front, and decreases below $c - \alpha$ to the right of it.

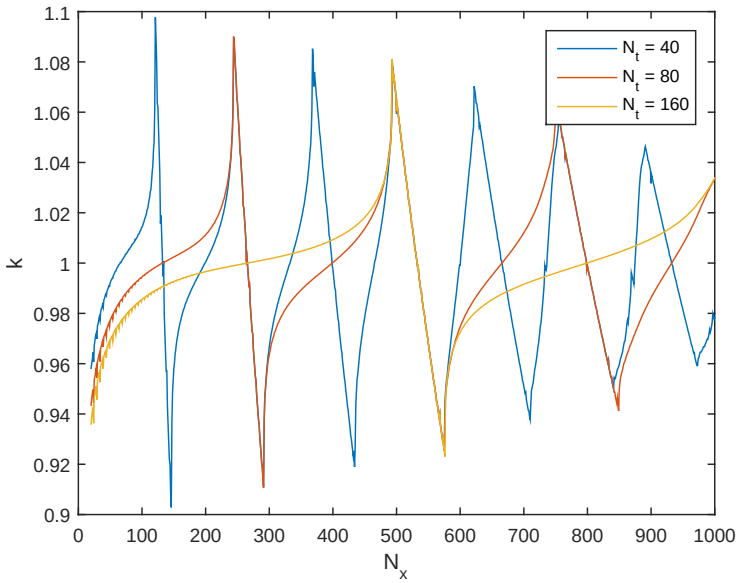

Figure 3-13: Calculated $\widehat{c}$, as in Figure 3-10, but with cubic Lagrange interpolation.

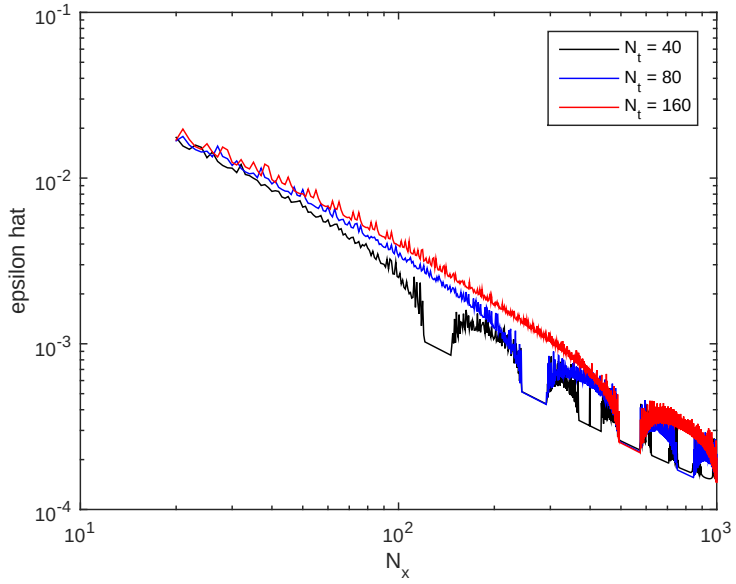

Figure 3-14: Calculated $\widehat{\varepsilon}$, using a flux limited cubic Lagrange interpolant, as in equations (2.48–2.51).

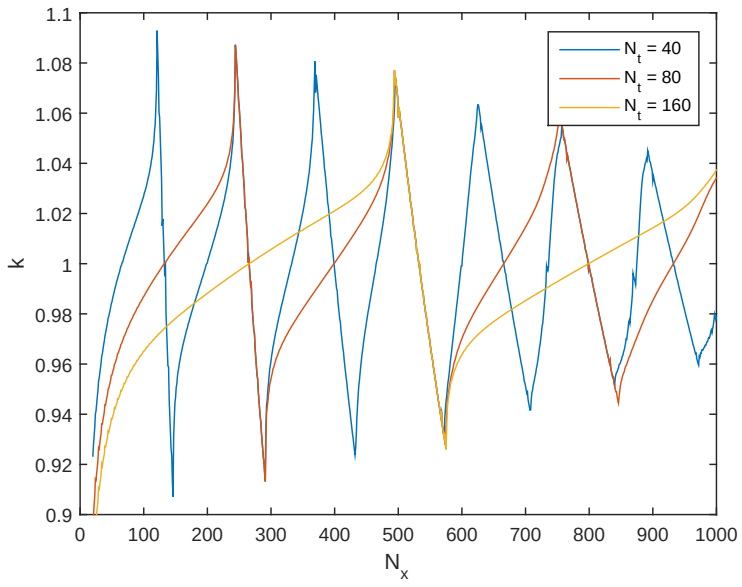

Figure 3-15: Calculated $\widehat{c}$, using a flux limited cubic Lagrange interpolant. Speeds are very similar to the case where no flux limiter is used.

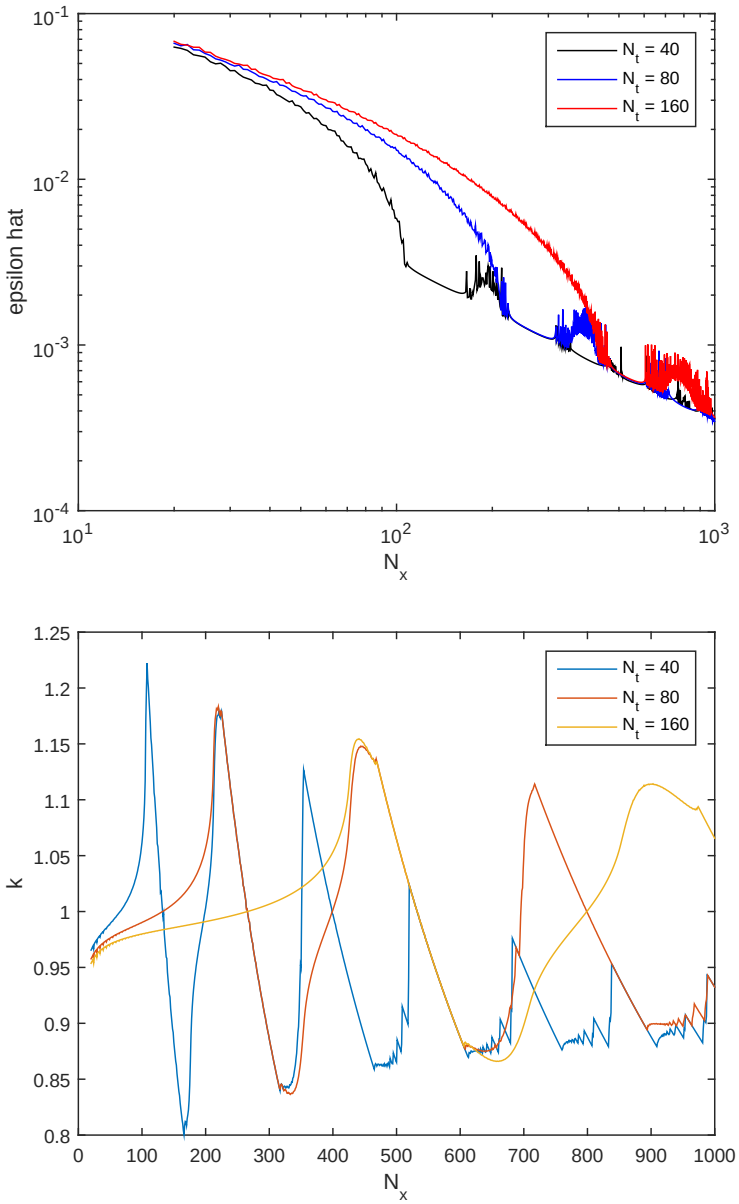

Figure 3-16: Numerical viscosity $\widehat{\varepsilon}$ and wave speed $\widehat{c}$ for $c = 1$, $\alpha = 0.25$ using linear interpolation. The results are qualitatively similar to those presented above for $\alpha = 0.1$, but with $\widehat{c}$ consistently slower than the actual speed $c = 1$ as N_x increases.

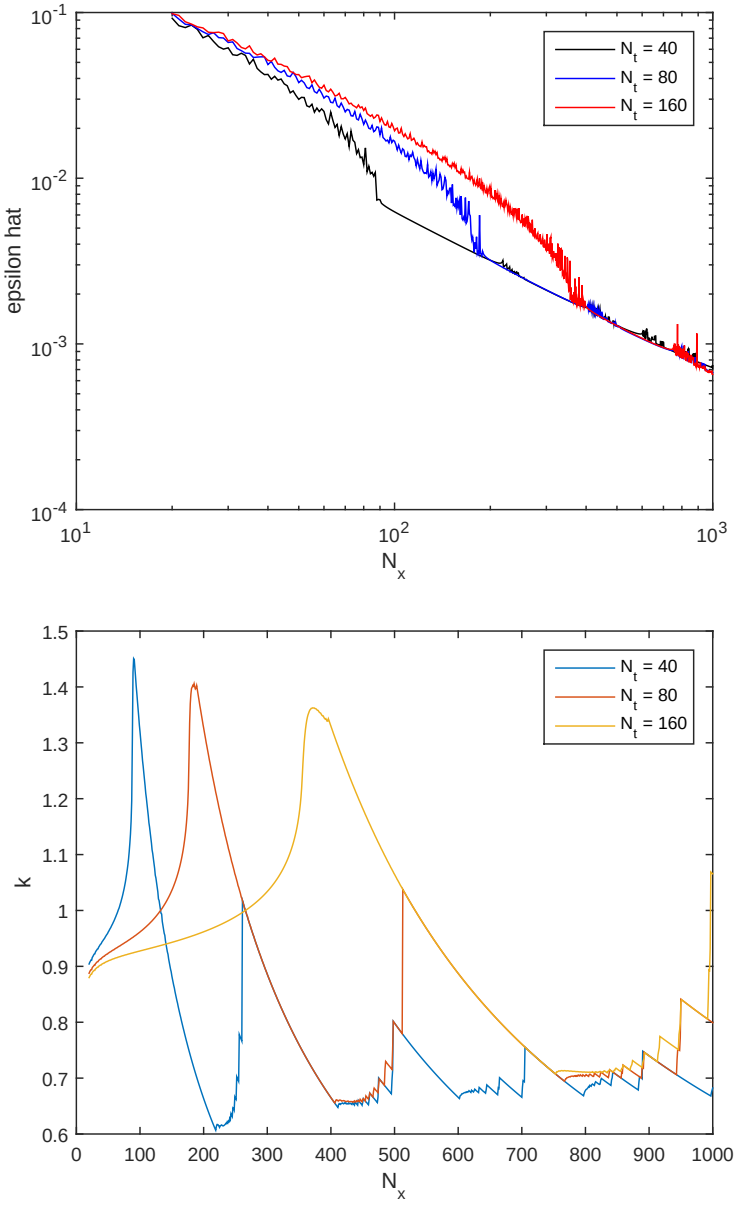

Figure 3-17: Numerical viscosity $\widehat{\varepsilon}$ and wave speed $\widehat{c}$ for $c = 1$, $\alpha = 0.5$ using linear interpolation. The numerical viscosity $\widehat{\varepsilon}$ is dominated by the aliasing limit and the speed is exhibiting drastically different behaviour, possibly due to the breakdown in the assumption that $U \approx c$ throughout the spatial domain.

3.6 Conclusions

We have analysed the SL discretisation of Burgers' equation for travelling wave solutions, including the discretisation of the kinematic equation and the linear interpolation, arriving at expressions for $\widehat{\varepsilon}$ and $\widehat{c}$. We have shown good agreement between the theory and numerical results, and have demonstrated that they exhibit similar behaviour when using off-centring and cubic-Lagrange interpolation.

Chapter 4

Review of Moving Mesh Methods

As observed in the previous chapter, the error in the SL method is largely dependent on Δx. In addition to this, the departure point error, interpolation error and finite difference error can all be shown to be small away from the front. It is thus desirable to have a high density of mesh points near to the front, and for this high density to track its movement as it advances. This motivates the use of adaptive moving meshes.

In this section we describe monitor function based techniques for moving a mesh in 1D, reducing the mesh width Δx locally to reduce errors. We then extend these to 2D problems by using a 1+1D mesh in (x, z), where the horizontal mesh width Δx is fixed, but the vertical mesh width in Δz can change.

4.1 Maps and Equidistribution

In this section we will show how the use of a monitor function can generate a mesh which equidistributes that function. The size of the monitor function controls the mesh in a regular manner.

4.1.1 Maps and Monitor Functions

In the problems considered here, we hope to resolve issues which arise from numerical errors being accumulated in a limited localised area. In such situations the error could be reduced by having more mesh points in these areas.

This section is motivated by the idea of mesh densities specifying areas where we want more mesh points. This will be developed from the idea of maps.

We consider a bijective mapping between two 1D domains, a *computational domain* Ω_c and a *physical domain* Ω_p such that the image of any mesh in Ω_c is a mesh in Ω_p, retaining the same number of points and the same ordering.

As an example, we consider expressing a mesh in a physical domain $\Omega_p = [a, b]$ with a computation domain $\Omega_c = [0, 1]$ and a map

$$x : \Omega_p \to \Omega_c \tag{4.1}$$

$$\xi \mapsto x(\xi)\,, \tag{4.2}$$

which satisfies

$$x(0) = a\,, \quad x(1) = b\,, \quad \text{and} \quad \frac{\mathrm{d}x}{\mathrm{d}\xi}(\xi) > 0 \quad \forall \xi \in [0, 1]\,. \tag{4.3}$$

If we consider a uniform mesh in Ω_c

$$\xi_i = \frac{i}{N}\,, \quad i = 0, 1, \ldots, N\,, \tag{4.4}$$

then unless $x(\xi)$ is linear, we can define a non-uniform mesh in Ω_p as

$$x_i = x(\xi_i)\,, \quad i = 0, 1, \ldots, N\,, \tag{4.5}$$

with

$$a = x_0 < x_1 < \ldots < x_N = b\,. \tag{4.6}$$

The mesh spacing $x_{i+1} - x_i$ is small when $\frac{\mathrm{d}x}{\mathrm{d}\xi}$ is small.

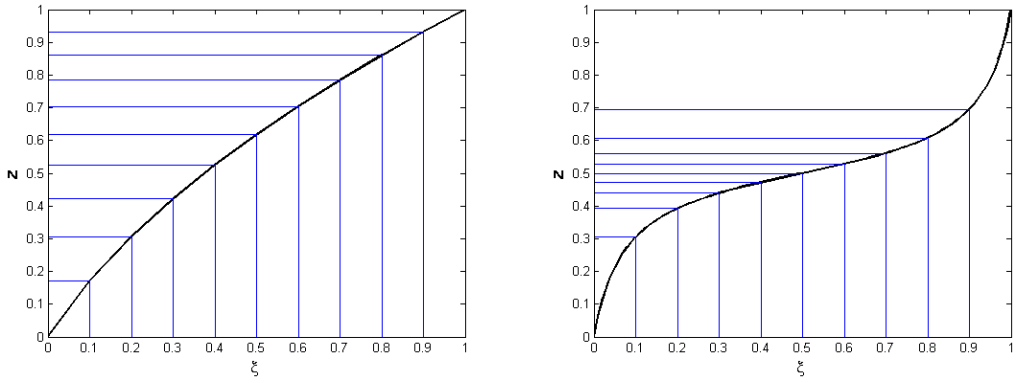

Figure 4-1: Some monotonic continuous maps with positive derivatives. The horizontal and vertical lines show the image of a uniform mesh.

Any such map has a corresponding positive *monitor functions*

$$M : \Omega_p \to \mathbb{R}^+\,, \tag{4.7}$$

which, as we shall see below, is inversely proportional to the gradient of x and hence the mesh spacing. We call a monitor function which has been normalised over Ω_p a *mesh density function* (MDF), denoted by ρ_M.

A monitor function M corresponding to x, integrated over some volume can be thought of as telling us what proportion of Ω_c will be mapped into that area. The integral of $M(x)$ over the image of two equal intervals under x will be equal. If we again consider a map $x(\xi)$ from $\Omega_c = [0, 1]$ onto $\Omega_p = [a, b]$, and $\xi_1, \xi_2 \in \Omega_c$, then this relationship between M and x can be expressed as

$$\int_{x(\xi_1)}^{x(\xi_2)} M(x')\,\mathrm{d}x' = (\xi_2 - \xi_1)\theta_M\,, \tag{4.8}$$

where

$$\theta_M = \int_a^b M(x')\,\mathrm{d}x'\,. \tag{4.9}$$

Alternatively this can be expressed as

$$\int_a^{x(\xi)} M(x')\,\mathrm{d}x' = \xi\theta_M\,, \tag{4.10}$$

or for a mesh density function

$$\int_a^{x(\xi)} \rho_M(x')\,\mathrm{d}x' = \xi\,. \tag{4.11}$$

We can differentiate equation (4.10) to get a more direct relationship between the map and the monitor function

$$\frac{\mathrm{d}\xi}{\mathrm{d}x} = \frac{1}{\theta_M} M(x)\,. \tag{4.12}$$

This is the 1D equidistribution equation. If this equation is satisfied, then we say that x *equidistributes* M.

In Figure 4-1 we see two such maps which have associated mesh densities seen in Figure 4-2. The thin lines and crosses have been added to show the link with meshes; In each case the image of a uniform mesh is shown, partitioning Ω_p into intervals on which the integral of M is constant.

We will use the principle of equidistribution (4.12) to place our mesh points in our physical domain to have a desired distribution. This is useful when solving PDEs numerically, where we may want to spread some numerical error evenly across our domain, e.g. interpolation or discretisation error. The general idea is that if the numerical error is high in a region, we can try to locally reduce this error by increasing the mesh density function, and hence the number of mesh points present in that region.

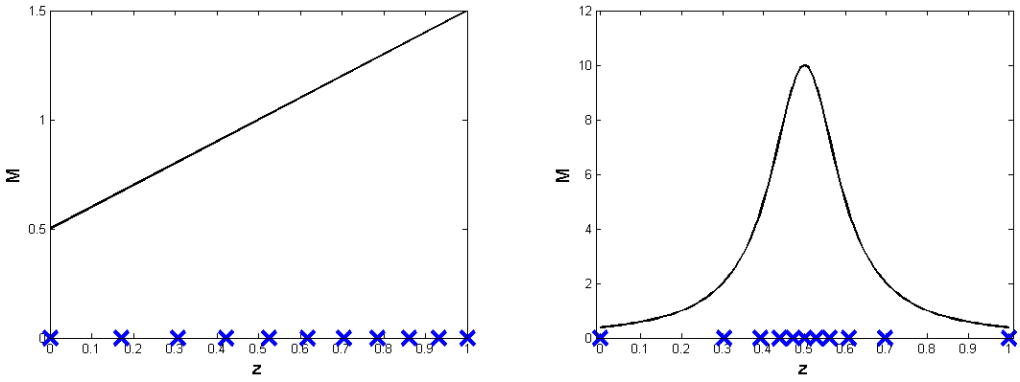

Figure 4-2: Some MDFs corresponding to the maps in Figure 4-1. The crosses represent the image of a uniform mesh in Ω_c under x, and hence the integral between all pairs of adjacent crosses is equal.

For now we introduce some idealised monitor functions to help look at how equidistribution behaves when calculated numerically.

4.1.2 Monitor Functions Used

In problems where errors are localised in a small area, we want a monitor function which is large locally. An example of such a function is the so called Witch of Agnesi,

$$M_\epsilon(x) = \frac{\epsilon}{\epsilon^2 + x^2}\,, \tag{4.13}$$

for a small parameter $\epsilon > 0$. The Witch of Agnesi makes a useful monitor function, since it can be used to approximate singularities in blow up problems (2D example given in [6, §5 Ex. 6]): $M_\epsilon(x)$ has an integral over $\mathbb{R}$ of π and takes the value $M_\epsilon(0) = \epsilon^{-1}$, with the majority of its density concentrated with a width of order ϵ centred on $x = 0$. In the limit of $\epsilon \to 0$, we get

$$\lim_{\epsilon \to 0} \left(M_\epsilon(x)\right) = \pi\delta(x)\,, \tag{4.14}$$

where $\delta(x)$ is the Kronecker delta function.

The Witch of Agnesi (4.13) has an integral

$$\int M_\epsilon(x)\mathrm{d}x = \arctan\left(\frac{x}{\epsilon}\right) + \text{constant}\,, \tag{4.15}$$

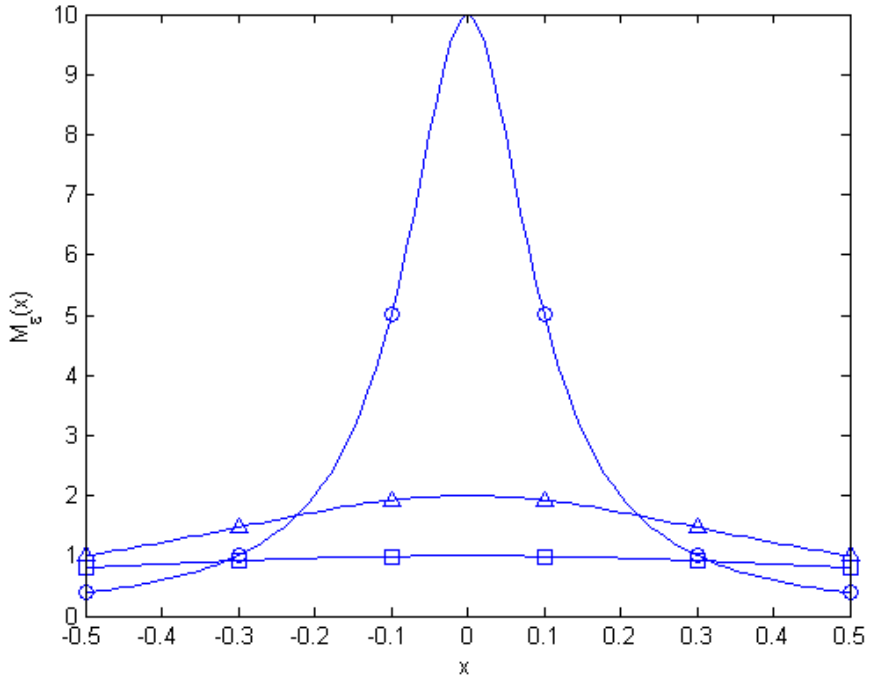

Figure 4-3: $M_\epsilon(x)$ (equation (4.13), Witch of Agnesi) for different values of ϵ; 0.1 (circles), 0.5 (triangles) and 1 (squares).

and we can use the equidistribution equation (4.10) to get

$$\begin{aligned} \theta_M \xi &= \int_a^{x(\xi)} M_\epsilon(x')\mathrm{d}x' \\ &= \arctan\left(\frac{x}{\epsilon}\right) - \arctan\left(\frac{a}{\epsilon}\right) . \end{aligned} \tag{4.16}$$

This can be rearranged to give an explicit expression for the corresponding map $x(\xi)$, as

$$x(\xi) = \epsilon \tan\left(\theta_M \xi + \arctan\left(\frac{a}{\epsilon}\right)\right) . \tag{4.17}$$

Later, when using these methods to solve PDEs we use a solution dependent monitor function, modified arc-length, or

$$M_u(x) = \sqrt{b + (u_x)^2}\,, \tag{4.18}$$

for some parameter b giving a high density of mesh points around areas of high gradient. In the case of $b = 1$ $M_u(x)$ gives the arc-length, and would distribute points along the curve $u(x)$. This is particularly useful for piecewise constant interpolation.

A curvature-based monitor function

$$M_u(x) = \sqrt{b + (u_{xx})^2} \tag{4.19}$$

would lead to a high density of mesh points about areas where u_{xx} is high. This is useful for reducing the error of a piecewise linear interpolant, since the linear interpolation error (see equation (2.35) in Section 2.5) depends on the second derivative.

In the following sections we introduce two methods for numerically finding the set of points that equidistribute a given monitor function.

4.1.3 Trapezium Equidistribution

As described above, we are looking for the points x_i, $i = 1, 2, \ldots, N$, which equidistribute the monitor function $M(x)$ on $[a, b]$. For the Witch of Agnesi monitor function, we were able to use the equidistribution equation to find an explicit expression for $x(\xi)$. In general finding such an expression may not be possible. If $M(x)$ is integrable such that

$$\int_a^x M(x')\mathrm{d}x' = \theta_M \xi(x)\,, \tag{4.20}$$

but the resulting function $\xi(x)$ cannot be rearranged to give an explicit expression for x, then we can numerically find the equidistributing x_i by finding the roots of

$$\theta_M \xi(x_i) - \frac{i\theta_M}{N} = 0\,, \quad i = 1, \ldots, N\,. \tag{4.21}$$

This could be done using Newton's method; given $x^{(k)}$, an estimate to x_i, we can find $x^{(k+1)}$ as

$$x^{(k+1)} = x^{(k)} - \frac{1}{M(x^{(k)})}\left(\theta_M \xi(x^{(k)}) - \frac{i\theta_M}{N}\right)\,. \tag{4.22}$$

If the integral of $M(x)$ has no closed form, we can approximate $M(x)$ by another function, for example by a piecewise polynomial. Care must be taken to ensure that any approximating function is strictly positive. If the approximating function is a piecewise polynomial, it is sufficient to use a monotonic interpolating polynomial, such as those discussed in Section 2.5.4.

A cheap and effective method for equidistributing for a discretely defined monitor function, is to represent M by a piecewise linear function. It is then straightforward to find the x_i which equidistribute this approximation. This method has been implemented in the code `equidistribute.m` in Appendix A.2.

If $M(x)$ is available for all x in $[a, b]$, then we could evaluate at N points, find the x_i, $i = 1, 2, \ldots, N$ which equidistribute the linear interpolant of $M(x)$, then repeat

with the linear interpolant over the x_i. This process is not recommended since there are situations where this process causes mesh points to oscillate around the desired mesh, as in Figure 4-8. This approach could be dampened by taking a relaxation of these meshes, i.e. when a new mesh is attained, average it with the previous mesh, but it would be more beneficial to sample at more points and equidistribute to a more accurate approximation to $M(x)$.

A potential extension would be to take a polynomial interpolating function of higher order and calculate exact equidistribution to this interpolant, but care would have to be taken to ensure the interpolant of $M(x)$ is strictly positive.

4.1.4 Equidistribution with MMPDEs

We consider a moving mesh PDE (MMPDE), where the movement is determined by an elliptic time dependent PDE

$$x_t = \frac{1}{\tau}(Mx_\xi)_\xi \,, \tag{4.23}$$

known as MMPDE5 [19]. This has a steady state solution when equidistribution (4.12) is satisfied, towards which x evolves [6]. We can use this equation (4.23) to find equidistribution of points numerically for a given M by integrating it forward in time. There are a number of other MMPDEs, some of which are described in [20].

Here there are two paths we can consider for evolving a mesh in a time dependent PDE. One is to keep a solution constant and adapt in pseudo-time, either for a set amount of time or until we are within tolerance of equidistribution as for trapezium equidistribution.

The second is to solve (4.23) dynamically with the PDE, where we couple this MMPDE with another time dependent PDE in transformed variables to get a moving mesh solution to that PDE. The monitor function may depend on the solution of the PDE with a view to increasing mesh resolution in regions deemed important, possibly across a meteorological front where truncation error may be high. Figure 4-4 shows Burgers' equation solved with MMPDE5, as in [21]. This experiment is described in detail in Chapter 5.

4.2 Mesh calculations for meteorogically relevant geometries.

At the Met Office, much of the parametrisation relies on computational cells being arranged in vertical columns, and all of the meshes in the dynamics calculations have

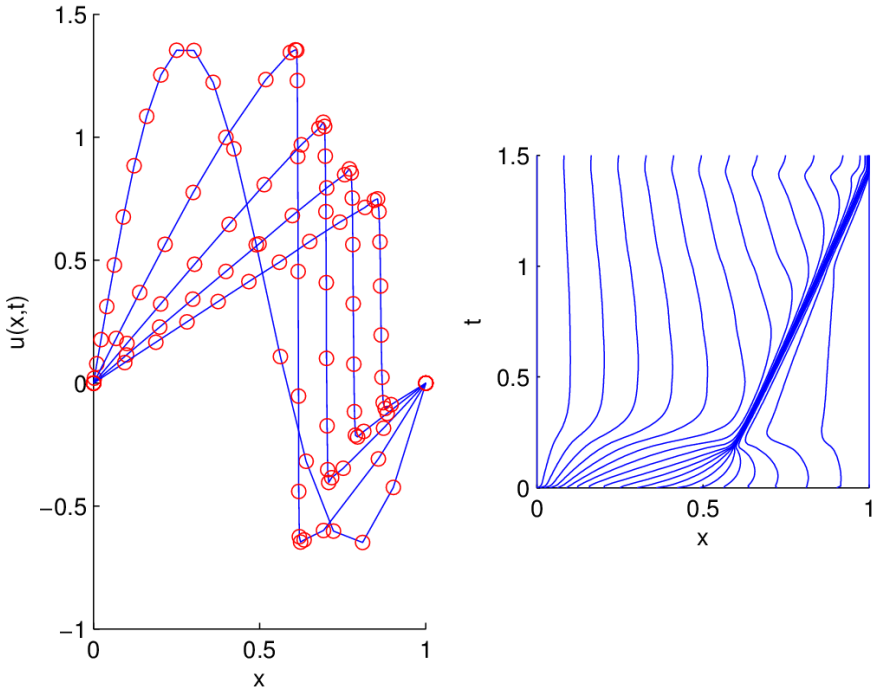

Figure 4-4: Burgers' equation solved with moving mesh PDE. Initial conditions are given as the sine curve, then the solution is shown at various points up to $t = 1.5s$. The right image shows the movement of the mesh points with time.

this property. As a result of this, any vertical variation in the mesh will be decoupled from horizontal variation, although different columns can have different meshes. We call 2D meshes with this configuration $1 + 1$D meshes, and we call a 3D mesh with strictly vertical columns a $1 + 2$D mesh.

Here we look at some of the ways we can numerically find a mesh which equidistributes vertically and some horizontal-vertical slice meshes that result from these calculations. We also look at some additional desirable properties that we can impose on the mesh by modifying any monitor function.

We take an x-z slice through a domain with a gentle hill (smooth orography, here a sinusoidal curve), with terrain following coordinates.

We then assign a temperature to every point in our domain with an "inversion layer", a line cutting the domain in two, represented in Figure 4-5 by the thick red line with a high temperature above and a low temperature below, changing continuously with a width parameter w. The temperature is taken to be

$$\theta(z) = \tanh\left(\frac{z - p(x)}{w}\right), \tag{4.24}$$

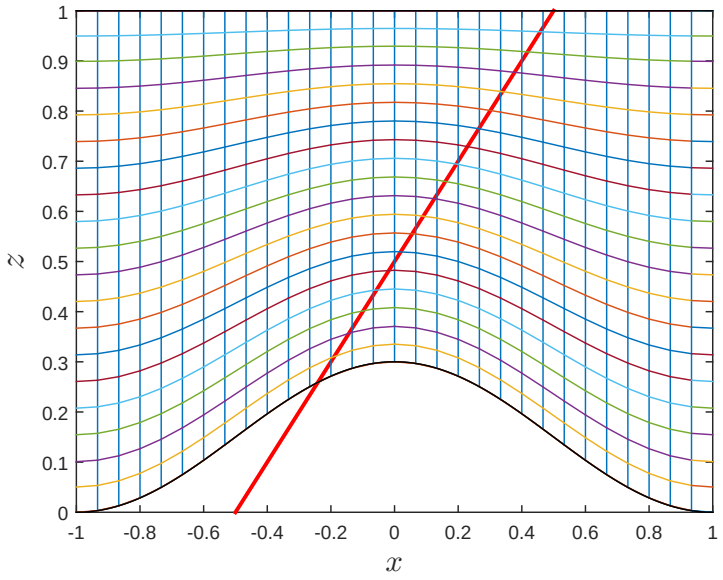

Figure 4-5: A contrived model of an inversion layer; a sharp temperature change at the interface of two air masses, the interface represented by the red line intersecting a sinusoidal hill. The mesh here is based only on the orography.

where $z = p(x)$ is the inversion layer.

We can now perform 1D adaptation purely in the z direction with fixed x, leading to a $1 + 1$D mesh. We use vertical arc-length from (4.18) along the temperature for our monitor function,

$$M(z) = \sqrt{1 + (\theta_z)^2}\,. \tag{4.25}$$

Taking the cubic spline of this data at the initial mesh points (using the MATLAB implementation of cubic spline) then we find the mesh which equidistributes this function. We see the result of this adaptation in Figure 4-6.

Note in particular the large compression/expansion of mesh points where the inversion enters and exits the domain. This is due to the sudden jump in $\theta_M(x) = \int M(x, z)\mathrm{d}z$. For the remainder of this section we look at techniques for $1+1$D meshes.

This particular method of equidistributing to the spline can lead to issues when there are big jumps in the monitor function, as we see in Figure 4-7. The spline has under-shot to capture the jump in M, leading to a grid with areas of negative density. This is easily remedied by using a monotonic interpolant, as in Section 2.5.4.

We look at how we equidistribute numerically in more depth in the next section.

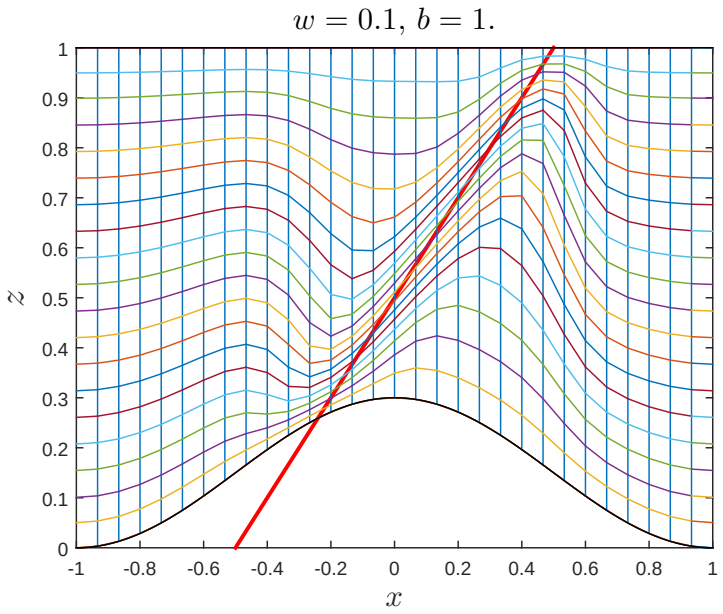

Figure 4-6: The resulting mesh after equidistributing vertically with $M = \sqrt{b + (\theta_z)^2}$. The width of the inversion is of order w.

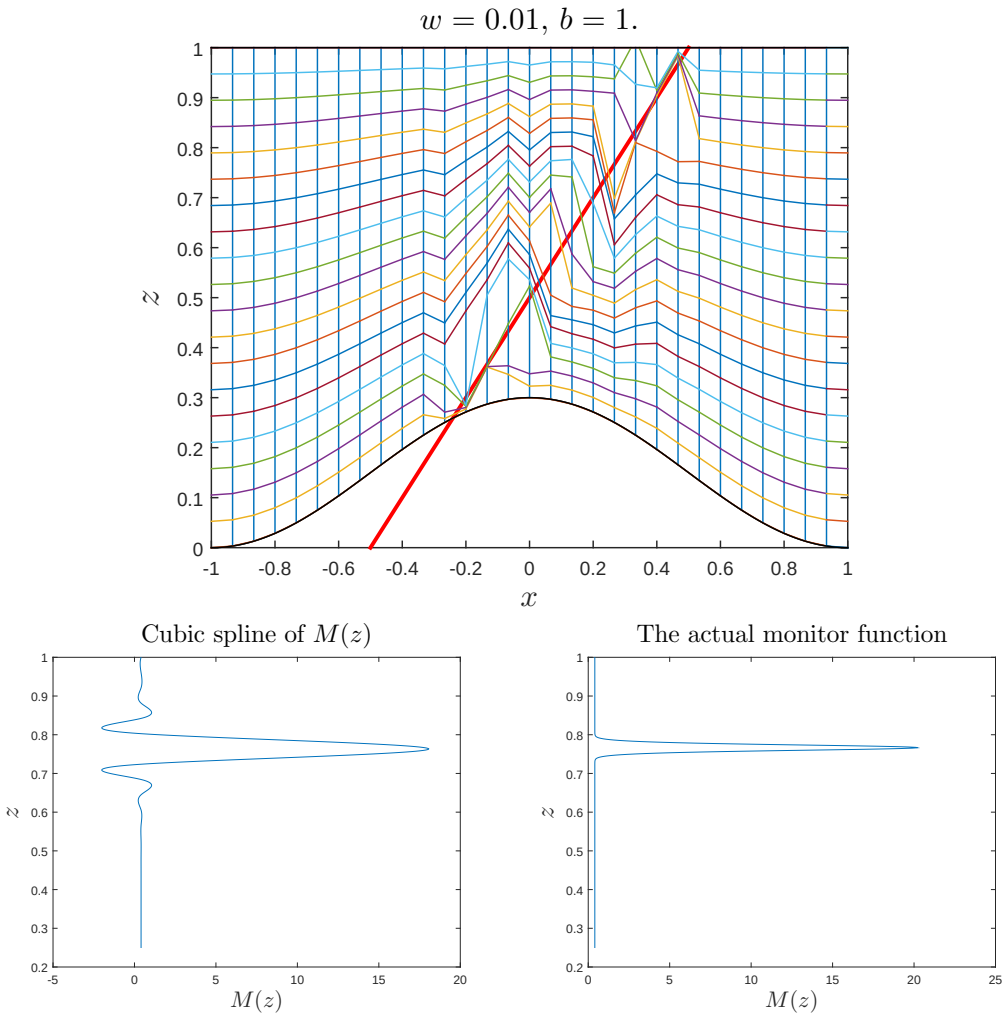

Figure 4-7: Showing the dangers of using a cubic spline as a monitor function. The lower pictures show the cubic spline used when the mesh leaves the domain (at $x \approx 0.25$) and the actual (positive) monitor function.

4.2.1 Time-independent Monitor Functions

We consider an integrable monitor function to observe convergence of numerical equidistribution.

For given maps and corresponding monitor functions, we can numerically solve these via the methods described in Sections 4.1.3 and 4.1.4 and look at the convergence towards equidistribution.

We consider 1D maps with monitor functions of the form

$$M_{p,\epsilon}(z) = \frac{1}{\pi}\frac{\epsilon}{\epsilon^2 + (z-p)^2}\,. \tag{4.26}$$

This is chosen as an idealised inversion layer centred on p and changing over a width of an order of magnitude ϵ.

We consider iterated trapezium equidistribution, where the monitor function is resampled after each call to `equidistribute.m`, and MMPDE5 solved with the MATLAB stiff ODE solver ODE15s.

Here we aim to to present trapezium equidistribution as an evolution operator with fixed time step, attempting to solve directly in one step and try to get a comparable relaxation form of this.

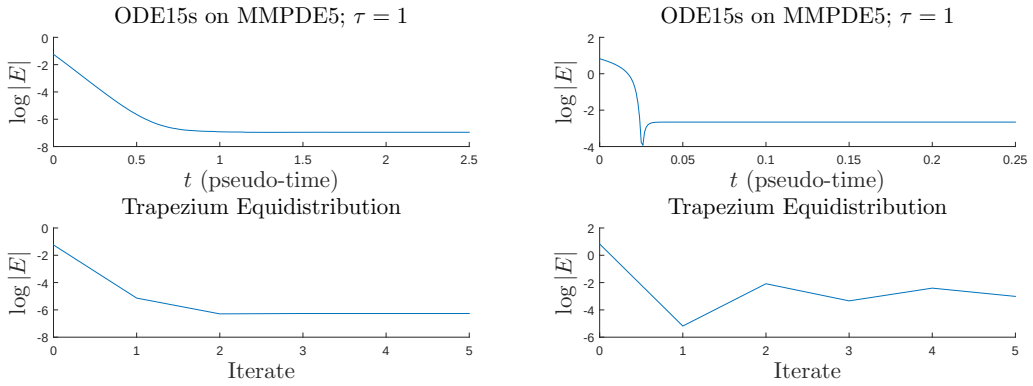

Figure 4-8: Showing the convergence of monitor functions and methods with the 2-norm of the error from exact equidistribution. Equidistributing 11 points across the interval $[0, 1]$ using a first-order finite difference scheme of MMPDE5 and iterated equidistribution of piecewise linear approximation of M. Left plots show the monitor function $M(x) = 1/\sqrt{2x + 1/4}$, and right plots show the monitor function $M_\epsilon(x) = \epsilon/(\epsilon^2 + (x - 1/2)^2)$, $\epsilon = 0.01$ (trapezium equidistribution does not converge).

In Figure 4-8 we see that with the exception of the iterated trapezium equidistribution and the Witch of Agnesi these methods reach a point where we have a numerical steady state and the equidistribution "distance" measure is zero, i.e. the discretised

form of $(Mx_\xi)_\xi = 0$.

We have reached equidistribution relative to our discretisation; the remaining error is discretisation error. We observe the convergence of this discretisation error with Figure 4-9.

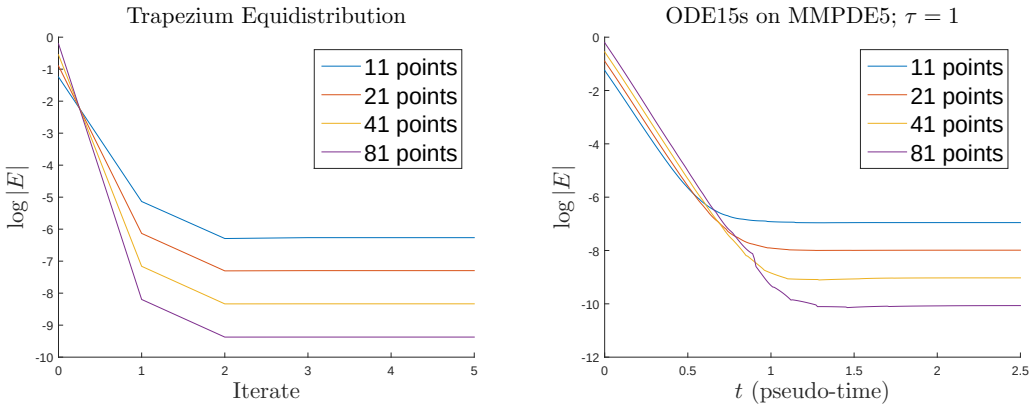

Figure 4-9: Convergence of methods for monitor function $1/\sqrt{2x + 1/4}$ for (decreasing steady state error) 11, 22, 44 and 88 grid points.

4.2.2 Element Quality

Once we have performed this 1D adaptation, we want some tools to look at the cells created in the $1 + 1$D vertical slice mesh.

Taken from [16] and [44] we have some quadrilateral measures whereby any quadrilateral in an x-z plane can be characterised by area, x-tapering, z-tapering, skewness and aspect-ratio, with location and orientation fixing the remaining three degrees of freedom. Examples of these properties can be seen in Figure 4-10.

In our case we are currently just considering z-adaptation, so we can consider the orientation of the elements aligned with x constant, hence also no x-tapering.

These measures are calculated in the code included with [16] with minor alterations. Colour-gradient plots are shown in Figure 4-11. Such measures could be employed to analyse resulting $1 + 1$D meshes, though care must be taken not attribute too much importance to a single measure, as a "bad" mesh for one application might be considered a "good" mesh for another.

4.2.3 Averaging

For a given monitor function we can impose a maximum grid-width by averaging the monitor function with a constant.

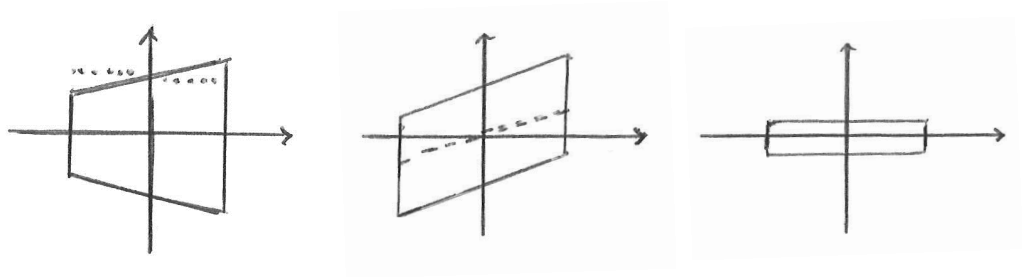

Figure 4-10: Visual example of Robinson quadrilateral measures showing, from left to right: z-tapering, no skewness; skewness, no tapering; high aspect ratio, no skewness or tapering.

To see the effects of averaging we consider N points uniformly distributed over $[0, \epsilon]$, $x_0 = 0$, $x_{N-1} = \epsilon$, with $x_N = 1$.

A corresponding monitor function can be given as

$$M(x) = \begin{cases} \dfrac{N-1}{N\epsilon}\,, & 0 \le x < \epsilon \\ \dfrac{1}{N(1-\epsilon)}\,, & \epsilon \le x \le 1\,, \end{cases} \tag{4.27}$$

with $\theta_M := \int_0^1 M(x)\mathrm{d}x = 1$. The averaged monitor function, with equal weighting is

$$\begin{aligned} \hat{M}(x) :&= \frac{M(x) + \theta_M}{2\theta_M} \\ &= \begin{cases} \dfrac{(1+\epsilon)N - 1}{2N\epsilon}\,, & 0 \le x < \epsilon \\ \dfrac{(1-\epsilon)N + 1}{2N(1-\epsilon)}\,, & \epsilon \le x \le 1\,. \end{cases} \end{aligned} \tag{4.28}$$

Since

$$\frac{(1-\epsilon)N+1}{2N(1-\epsilon)}\,\frac{(1-\epsilon)N+1}{2N(1-\epsilon)+2} = \frac{1}{2}\,, \tag{4.29}$$

the integral of $\hat{M}$ over $[\epsilon, 1]$ is greater than $\frac{1-\epsilon}{2}$, and approximately half the equidistributed points lie in the region $[\epsilon, 1]$, which previously had only the endpoints ϵ and 1.

A similar technique is used in [34], where the authors use a regularisation (equivalent to the arc-length equation (4.18) here) of a monitor function $M(x)$ as

$$\hat{M} = \sqrt{1 + c^2 M^2}\,, \tag{4.30}$$

where the two values of parameter used in the results section are $c = 0.5$ and $c = 0.25$.

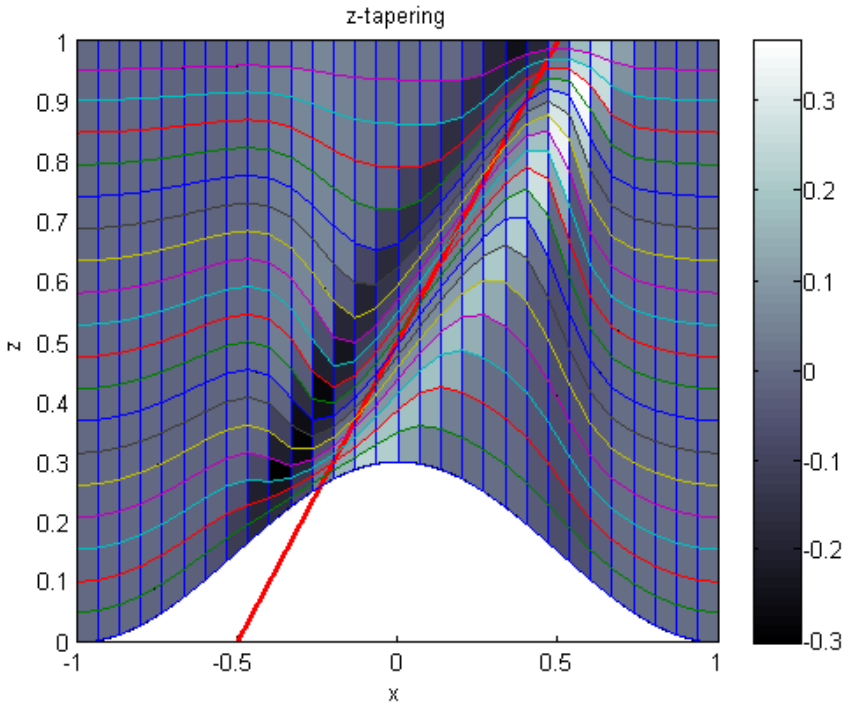

Figure 4-11: Plots showing z-tapering of the elements of the mesh from Figure 4-6.

4.2.4 Smoothing

Another property we may wish to impose is mesh smoothness, where there are no sudden jumps from areas of high mesh-width to low mesh-width. This can be achieved through smoothing of the monitor function.

We can consider smoothing of the monitor function over grid points, for example in 1D, a three-point smoothing of f at points x_i, $i = 0, .., N$ could be

$$\mathcal{S}[f(x_i)] = \frac{(\alpha_{-1} f(x_{i-1}) + \alpha_0 f(x_i) + \alpha_1 f(x_{i+1}))}{\alpha_{-1} + \alpha_0 + \alpha_1}, \tag{4.31}$$

for internal points $i = 1, \ldots, N-1$, and

$$\mathcal{S}[f(x_0)] = \frac{(\alpha_0 f(x_0) + \alpha_1 f(x_1))}{\alpha_0 + \alpha_1}, \tag{4.32}$$

$$\mathcal{S}[f(x_N)] = \frac{(\alpha_{-1} f(x_{N-1}) + \alpha_0 f(x_N))}{\alpha_{-1} + \alpha_0}. \tag{4.33}$$

This smoothing operation can be expressed as a smoothing stencil

$$\boxed{\begin{array}{ccc} \alpha_{-1} & \alpha_0 & \alpha_1 \end{array}}$$

or, if a vector $\mathbf{f}$ is composed with $f_i = f(x_{i-1})$ then the effect of smoothing can be expressed by left-multiplying by a tridiagonal matrix S, where

$$S = \begin{pmatrix} \frac{\alpha_0}{\alpha_0+\alpha_1} & \frac{\alpha_1}{\alpha_0+\alpha_1} & & & \\ \frac{\alpha_{-1}}{\alpha_{-1}+\alpha_0+\alpha_1} & \frac{\alpha_0}{\alpha_{-1}+\alpha_0+\alpha_1} & \frac{\alpha_1}{\alpha_{-1}+\alpha_0+\alpha_1} & & \\ & \ddots & \ddots & \ddots & \\ & & \frac{\alpha_{-1}}{\alpha_{-1}+\alpha_0+\alpha_1} & \frac{\alpha_0}{\alpha_{-1}+\alpha_0+\alpha_1} & \frac{\alpha_1}{\alpha_{-1}+\alpha_0+\alpha_1} \\ & & & \frac{\alpha_{-1}}{\alpha_{-1}+\alpha_0} & \frac{\alpha_0}{\alpha_{-1}+\alpha_0} \end{pmatrix}. \tag{4.34}$$

If we have a 2D domain and we define the N-by-N matrix F such that

$$F_{i,j} = f(x_i, y_j)\,, \tag{4.35}$$

then smoothing in the x-direction can be effected by left-multiplication with S, i.e.

$$\mathcal{S}_x[F] = SF\,, \tag{4.36}$$

and smoothing in the y-direction by right-multiplication with the transpose of S,

$$\mathcal{S}_y[F] = FS^T\,. \tag{4.37}$$

These smoothing operations are associative. As an example, using the stencil

$$\boxed{\begin{array}{ccc} 1 & 2 & 1 \end{array}}$$

for both x and y smoothing is equivalent to the 2D smoothing stencil

$$\boxed{\begin{array}{ccc} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{array}}.$$

In [34], a horizontal smoothing step is used with a stencil

$$\boxed{\begin{array}{ccc} 0 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 0 \end{array}},$$

which is iterated a number of times to "reduce the loss of horizontal coherence". In the two experiments presented for Data Assimilation, the smoothing stencil is applied 6

and 8 times, equivalent to a single 13-by-13 and 17-by-17 horizontal smoothing stencil, and was no vertical smoothing.

It is worth noting that in 1D this stencil smoothing operator can also be represented by a convolution of the sequence of points $\{x_i\}$ with $[..., 0, 0, \alpha_{-1}, \alpha_0, \alpha_1, 0, 0, \cdots]$, with appropriate definition of the sequences and treatment near the edges of the domain. For a non-discrete monitor function, an analogue, continuous form of smoothing is convolution with a function such as M_σ. If we consider such a convolution on the monitor function M_ϵ,

$$\bar{M}(x) = (M_\epsilon * M_\sigma)(x)\,, \tag{4.38}$$

and since

$$M_\epsilon(x) = F\left[\frac{1}{2}e^{-\epsilon|s|}\right]\,, \tag{4.39}$$

for F as the Fourier transform, we get

$$\bar{M}(x) = F\left[\frac{1}{4}e^{-(\epsilon+\sigma)|s|}\right]\,. \tag{4.40}$$

Chapter 5

Application of Moving Meshes to Burgers' Equation

In this chapter we combine the ideas and results of the previous two chapters in order to calculate the solution of Burgers' equation on a moving mesh. In the first part we will use an Eulerian (non-semi-Lagrangian) implementation, which can have issues with CFL conditions. In the second part we will combine the semi-Lagrangian method with moving meshes to give a very effective procedure for solving Burgers' equation.

5.1 Burgers' Equation

Having looked at mesh movement, we now turn our attention to using moving meshes to solve PDEs in one spatial dimension. As an example which is typical in the literature, we use Burgers' equation. In this section we look at Burgers' equation, finite difference discretisations on non-uniform meshes then show computations of moving and static mesh solutions to Burgers' equation with some simple error measures.

For some $\varepsilon > 0$, the viscous Burgers' equation as described in Section 2.6 is given by

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \varepsilon \frac{\partial^2 u}{\partial x^2}\,, \quad \forall x \in [x_L, x_R], \quad t > 0\,, \tag{5.1}$$

with initial and boundary conditions

$$\begin{aligned} u(x,0) &= u_0(x) \quad \forall x \in [x_L, x_R]\,, \\ u(x_L,t) &= u_L \quad \forall t > 0\,, \\ u(x_R,t) &= u_R \quad \forall t > 0\,. \end{aligned} \tag{5.2}$$

In this section we shall solve in a finite domain $[0, 1]$ with smooth initial conditions

$$u_0(x) = \sin(2\pi x) + \frac{1}{2}\sin(\pi x)\,, \tag{5.3}$$

which forms a sharp front in a short amount of time. Later in the chapter, we investigate the travelling wave solutions, as studied in Chapter (3).

We seek a numerical solution to Burgers' equation via the method of lines. On a mesh x_i, $i = 0, 1, \ldots, N+1$, our numerical solution $\mathbf{U}(t)$ approximates

$$U_i(t) \approx u(x_i, t)\,.$$

Approximating the spatial derivatives via finite difference leads to a system of $N+2$ ODEs which we integrate numerically.

5.2 Finite Differences on non-Uniform Meshes

We take a fixed mesh on the domain $[x_L, x_R]$ with $N+2$ points,

$$x_L = x_0 < x_1 < \ldots < x_N < x_{N+1} = x_R\,, \tag{5.4}$$

and define the differences

$$\Delta_i := x_i - x_{i-1} \quad \forall i = 1, \cdots, N+1\,. \tag{5.5}$$

For a uniform mesh we use the notation

$$\Delta x := \Delta_i \tag{5.6}$$

$$= \frac{x_R - x_L}{N+1}\,. \tag{5.7}$$

We adopt the common notation for finite difference operators as in Section 3.3 but extended to non-uniform meshes: for the finite difference approximation to $u_x(x_i, t)$ we use the forward difference, backwards difference and central difference operators. These we define as

$$\delta_x^+ U_i := \frac{U_{i+1} - U_i}{\Delta_{i+1}}\,, \tag{5.8}$$

$$\delta_x^- U_i := \frac{U_i - U_{i-1}}{\Delta_i}\,, \tag{5.9}$$

and

$$\delta_{2x}U_i := \frac{U_{i+1} - U_{i-1}}{\Delta_{i+1} + \Delta_i} \tag{5.10}$$

respectively. For u_{xx} we use the approximation

$$\delta_x^2 U_i := \frac{\delta_x^+ U_i - \delta_x^- U_i}{\frac{1}{2}(\Delta_i + \Delta_{i+1})} \tag{5.11}$$

which, for a uniform mesh, reduces to the more familiar definition

$$\delta_x^2 U_i = \frac{U_{i+1} - 2U_i + U_{i-1}}{(\Delta x)^2}\,. \tag{5.12}$$

We can also consider these finite difference operators acting on a vector $\mathbf{U}$. With an abuse of notation, δ_x^2 can be regarded as a matrix with action given by

$$\left(\delta_x^2 \mathbf{U}\right)_i = \delta_x^2 U_i\,. \tag{5.13}$$

In fact, the matrix is given explicitly for a non-uniform mesh in equation (5.54).

In this chapter we restrict ourselves to using Dirichlet boundary conditions, so that U_0 and U_{N+1} are fixed, and we reduce our problem to solving N ODEs, so $i = 1, \ldots, N$.

5.2.1 Truncation Error of Finite Difference Operators

Considering again the unstructured mesh, we want to look at the truncation error of our finite difference operators. For now we consider the central difference and the second order finite difference operators. Using the definitions of our operators, we can look at the errors of the central difference operator and the second difference operator, τ_{1i} and τ_{2i} respectively, that satisfy

$$\begin{aligned} \frac{\partial u}{\partial x}(x_i, t) &= \delta_{2x} U_i + \tau_{1i} \\ &= \frac{U_{i+1} - U_{i-1}}{\Delta_{i+1} + \Delta_i} + \tau_{1i}\,, \end{aligned} \tag{5.14}$$

$$\begin{aligned} \frac{\partial^2 u}{\partial x^2}(x_i, t) &= \delta_x^2 U_i + \tau_{2i} \\ &= \frac{\Delta_i U_{i+1} - (\Delta_i + \Delta_{i+1})U_i + \Delta_{i+1} U_i}{\frac{1}{2}\Delta_{i+1}\Delta_i(\Delta_{i+1} + \Delta_i)} + \tau_{2i}\,. \end{aligned} \tag{5.15}$$

We can express U_{i+1} and U_{i-1} as Taylor series expansions of $u(x)$ about x_i

$$U_{i+1} = u(x_i + \Delta_{i+1}) \tag{5.16}$$

$$= U_i + \Delta_{i+1}\frac{\partial u_i}{\partial x} + \frac{\Delta_{i+1}^2}{2}\frac{\partial^2 u_i}{\partial x^2} + \text{higher order terms (h.o.t)} \tag{5.17}$$

and

$$U_{i-1} = U_i - \Delta_i\frac{\partial u_i}{\partial x} + \frac{\Delta_i^2}{2}\frac{\partial^2 u_i}{\partial x^2} + \text{h.o.t}\,, \tag{5.18}$$

then substituting these terms into the finite difference approximations (5.14) and (5.15), we find the truncation errors τ_{1i}, τ_{2i} to leading order are

$$\tau_{1i} = -\left[\frac{\Delta_{i+1} - \Delta_i}{2}\frac{\partial^2 u_i}{\partial x^2} + \frac{{\Delta_{i+1}}^3 + {\Delta_i}^3}{6(\Delta_{i+1} + \Delta_i)}\frac{\partial^3 u_i}{\partial x^3}\right] + \text{h.o.t} \tag{5.19}$$

$$\tau_{2i} = -\left[\frac{\Delta_{i+1} - \Delta_i}{3}\frac{\partial^3 u_i}{\partial x^3} + \frac{{\Delta_{i+1}}^3 + {\Delta_i}^3}{12(\Delta_{i+1} + \Delta_i)}\frac{\partial^4 u_i}{\partial x^4}\right] + \text{h.o.t}\,. \tag{5.20}$$

Each τ_{ni} is made up of a mesh smoothness term $(\Delta_{i+1} - \Delta_i)$ and a mesh spacing term $\left(\frac{{\Delta_{i+1}}^3 + {\Delta_i}^3}{\Delta_{i+1} + \Delta_i}\right)$. On the uniform mesh the smoothness term vanishes and the spacing term becomes $(\Delta x)^2$, proportional to $(N+1)^{-2}$.

With motivation from the maps presented in Section 4.1, we note that we can consider x as a function of a computational variable $\xi \in [0,1]$. A mesh in the physical domain can be thought of the image of a uniform mesh in $[0,1]$ with

$$x_i = x(\xi_i)\,, \tag{5.21}$$

$$\xi_i := i\Delta\xi\,, \quad \forall i = 0, \cdots, N+1\,, \tag{5.22}$$

$$\Delta\xi := \frac{1}{N+1}\,. \tag{5.23}$$

If $x(\xi)$ is sufficiently smooth, then taking a Taylor series about $x(\xi_i)$, we see that

$$\Delta_i = x(\xi_i) - x(\xi_{i-1}) = x_i - \left(x_i - \Delta\xi\frac{\partial x_i}{\partial \xi} + \Delta\xi^2\frac{\partial^2 x_i}{\partial \xi^2} + \cdots\right)\,, \tag{5.24}$$

and we can express the truncation errors as

$$\tau_{1i} = -\left(\frac{1}{2}\frac{\partial^2 u_i}{\partial x^2}\frac{\partial^2 x_i}{\partial \xi^2} + \frac{1}{6}\frac{\partial^3 u_i}{\partial x^3}\left(\frac{\partial x_i}{\partial \xi}\right)^2\right)(\Delta\xi)^2 + \text{h.o.t}\,, \tag{5.25}$$

$$\tau_{2i} = -\left(\frac{1}{3}\frac{\partial^3 u_i}{\partial x^3}\frac{\partial^2 x_i}{\partial \xi^2} + \frac{1}{12}\frac{\partial^4 u_i}{\partial x^4}\left(\frac{\partial x_i}{\partial \xi}\right)^2\right)(\Delta\xi)^2 + \text{h.o.t}\,, \tag{5.26}$$

We observe that the two leading error terms have the same order of convergence, N^{-2}. These truncation errors can also be found in [5, Lemma 2.3 and 2.4].

In addition to the central difference approximation (5.14), we also consider a first order finite difference scheme

$$\frac{\partial u}{\partial x}(x_i, t) = \frac{\frac{1}{2}\Delta_i^2 U_{i+1} - \frac{1}{2}(\Delta_i^2 - \Delta_{i+1}^2)U_i - \frac{1}{2}\Delta_{i+1}^2 U_{i-1}}{\frac{1}{2}\Delta_{i+1}\Delta_i(\Delta_{i+1} + \Delta_i)} + \tau_{3i}\,, \tag{5.27}$$

with leading order truncation term

$$\tau_{3i} = \frac{\Delta_{i+1}\Delta_i}{6}\frac{\partial^3 u}{\partial x^3}(x_i, t) + \text{h.o.t}\,. \tag{5.28}$$

Comparing the truncation terms of (5.14) and (5.27), τ_{1i} and τ_{3i}, we observe that the latter does not have the same restrictions on mesh smoothness (manifest in the $(\Delta_{i+1} - \Delta_i)$ term in (5.25)), hence we would expect (5.27) to perform better on more general meshes. However, as we shall see in the next section, for Burgers' equation (5.14) is preferable since it leads to an energy preserving discretisation.

We use the second order finite difference scheme (5.15) for the viscosity term u_{xx}, and now proceed to discuss the remaining advection term.

5.2.2 Discretisation of the Nonlinear Advection Term

There are a number of options available for discretising the nonlinear advection term uu_x. In order to make an informed decision, we consider a discretisation of the inviscid Burgers' equation,

$$u_t + uu_x = 0\,, \quad x \in [x_L, x_R]\,. \tag{5.29}$$

We motivate our choice of discretisation of the nonlinear advection term by considering a discrete version of the energy.

An Energy Preserving Discretisation

Following [38] the energy associated with the inviscid Burgers' equation (5.29), considered over some spatial interval Ω, is defined as

$$E(t) = \int_\Omega \frac{u^2}{2} \mathrm{d}x \,. \tag{5.30}$$

Differentiating with respect to time and using inviscid Burgers' equation (5.29), provided we have a smooth solution u, we get

$$\begin{aligned} \frac{\mathrm{d}E}{\mathrm{d}t} &= \int_\Omega \frac{\partial}{\partial t}\left(\frac{u^2}{2}\right) \mathrm{d}x & (5.31)\\ &= \int_\Omega u u_t \mathrm{d}x = -\int_\Omega u^2 u_x \mathrm{d}x & (5.32)\\ &= \left.\frac{-u^3}{3}\right|_\Omega \,. & (5.33) \end{aligned}$$

We can interpret this as energy being conserved internally in the domain and only affected by the boundaries.

It is therefore desirable to have a discretisation which, in some sense, conserves the energy of the nonlinear term uu_x, and with this motivation we construct a discrete measure of energy as a numerical approximation to the integral (5.32),

$$\frac{\mathrm{d}E}{\mathrm{d}t} \approx \sum_i U_i \left(U_i\right)_t \frac{(\Delta_i + \Delta_{i+1})}{2} \,. \tag{5.34}$$

If we choose to discretise the nonlinear term as

$$U_i \left(U_i\right)_x = -U_i \delta_{2x} U_i \,, \tag{5.35}$$

then our approximation to the energy for inviscid Burgers' equation (5.29) is

$$\frac{\mathrm{d}E}{\mathrm{d}t} \approx \sum_i -U_i^2 \frac{U_{i+1} - U_{i-1}}{(\Delta_i + \Delta_{i+1})} \frac{\Delta_i + \Delta_{i+1}}{2} \,, \tag{5.36}$$

and considering only terms including U_j,

$$\begin{aligned} \frac{\mathrm{d}E}{\mathrm{d}t} &\approx \cdots - U_{j-1}^2 \frac{U_j - U_{j-2}}{2} - U_j^2 \frac{U_{j+1} - U_{j-1}}{2} - U_{j+1}^2 \frac{U_{j+2} - U_j}{2} - \cdots & (5.37)\\ &= -U_j \left(U_{j-1}^2 + U_j(U_{j+1} - U_{j-1}) + U_{j+1}^2\right) + \cdots \,. & (5.38) \end{aligned}$$

In general, these terms do not cancel out and in the sense of our approximation, energy is not conserved. This is also true with the second order finite difference discretisation (5.27).

Alternatively, with the choice for the nonlinear term uu_x as

$$-U_i \left(U_i\right)_x = -\frac{U_{i-1} + U_i + U_{i+1}}{3} \frac{U_{i+1} - U_{i-1}}{\Delta_i + \Delta_{i+1}}\,, \tag{5.39}$$

the discrete energies cancel out with successive terms and the total energy E is conserved. Therefore we elect to use the equation (5.39) as our discretisation. This is sometimes known as the FD2C discretisation of the nonlinear term. An alternative, referred to as the conservative form of the differential-difference equation [12, (4.109)], is

$$-U_i \left(U_i\right)_x = -\frac{1}{3} U_i \frac{U_{i+1} - U_{i-1}}{\Delta_i + \Delta_{i+1}} - \frac{1}{3} \frac{U_{i+1}^2 - U_{i-1}^2}{\Delta_i + \Delta_{i+1}}\,. \tag{5.40}$$

This discretisation also has the property of discrete energy conservation, but we shall not investigate it further.

We now return to the viscous Burgers' equation (5.1). The change in energy is

$$\begin{aligned} \frac{\mathrm{d}E}{\mathrm{d}t} &= \int_\Omega u u_t \,\mathrm{d}x \\ &= \int_\Omega u \left(-u u_x + \varepsilon u_{xx}\right) \,\mathrm{d}x \\ &= \left.\frac{-u^3}{3}\right|_\Omega + \varepsilon \left. u u_x \right|_\Omega \varepsilon \int_\Omega (u_x)^2 \,\mathrm{d}x \end{aligned} \tag{5.41}$$

$$,\,. \tag{5.42}$$

We observe that internally there is no contribution to the change in energy from the nonlinear advection term uu_x, only from the viscosity term εu_{xx}. Motivated by this, we shall use the FD2C discretisation (5.39) for the nonlinear advection term. For the viscosity term, we shall simply use the standard finite difference discretisation of the second derivative (5.15). Hence our discretisation of Burgers' equation is

$$\left(U_i\right)_t = -\frac{U_{i-1} + U_i + U_{i+1}}{3} \delta_{2x} U_i + \varepsilon \delta_x^2 U_i\,, \tag{5.43}$$

with an appropriate discretisation of the boundary conditions, which we discuss in the next section. Here we use the Dirichlet boundary conditions implemented as

$$U_0(t) := u(x_L, t)\,, \tag{5.44}$$

$$U_{N+1}(t) := u(x_R, t)\,, \tag{5.45}$$

and with the initial conditions

$$U_i(0) := u_0(X_i)\,, \tag{5.46}$$

we have a nonlinear system of ODEs which we can solve numerically.

We now give the numerical implementation of this system of equations (5.43–5.46).

5.2.3 Numerical Implementation and Boundary Conditions

As described above, we have $N+2$ points x_i, two of which correspond to the Dirichlet boundary conditions

$$u(x_0, t) = u_L\,, \tag{5.47}$$

$$u(x_{N+1}, t) = u_R\,. \tag{5.48}$$

This leaves N points on which to represent our solution $u(x_i, t)$. If we represent these values in a vector $\mathbf{U}$ with

$$U_i(t) \approx u(x_i, t)\,, \tag{5.49}$$

then we can express the central difference operator (e.g. $\delta_{2x}U_i$) from equation (5.14) as a corresponding N-by-N matrix (e.g. $\delta_{2x}\mathbf{U}$) and a length N boundary condition correction vector $\mathbf{b}_{\delta_{2x}}$, such that

$$\frac{\partial \mathbf{U}}{\partial x} \approx \delta_{2x}\mathbf{U} + \mathbf{b}_{\delta_{2x}}\,, \tag{5.50}$$

with

$$\delta_{2x} = \begin{pmatrix} 0 & \frac{1}{\Delta_1+\Delta_2} & & & \\ \frac{-1}{\Delta_2+\Delta_3} & 0 & \frac{-1}{\Delta_2+\Delta_3} & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & \frac{1}{\Delta_{N-1}+\Delta_N} \\ & & & \frac{1}{\Delta_N+\Delta_{N+1}} & 0 \end{pmatrix}, \tag{5.51}$$

$$\mathbf{b}_{\delta_{2x}} = \begin{pmatrix} \frac{-u_L}{\Delta_1+\Delta_2} & 0 & \cdots & 0 & \frac{u_R}{\Delta_N + \Delta N + 1} \end{pmatrix}^T. \tag{5.52}$$

The second order difference operator from equation (5.15) has a corresponding matrix

$$\frac{\partial^2 \mathbf{U}}{\partial x^2} \approx \delta_x^2\mathbf{U} + \mathbf{b}_{\delta_x^2}\,, \tag{5.53}$$

with

$$\delta_x^2 = \begin{pmatrix} \frac{-(\Delta_1+\Delta_2)}{d_1} & \frac{\Delta_1}{d_1} & & & \\ \frac{\Delta_3}{d_2} & \frac{-(\Delta_2+\Delta_3)}{d_2} & \frac{\Delta_2}{d_2} & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & \frac{\Delta_{N-1}}{d_{N-1}} \\ & & & \frac{\Delta_{N+1}}{d_N} & \frac{-(\Delta_N+\Delta_{N+1})}{d_N} \end{pmatrix}, \tag{5.54}$$

$$\mathbf{b}_{\delta_x^2} = \begin{pmatrix} \frac{\Delta_2 u_L}{d_1} & 0 & \cdots & 0 & \frac{\Delta_N u_R}{d_N} \end{pmatrix}^T, \tag{5.55}$$

where

$$d_i = \frac{1}{2}\Delta_i\Delta_{i+1}(\Delta_i + \Delta_{i+1}), \quad i = 1, \ldots, N. \tag{5.56}$$

Finally we can express the averaging operator from equation (5.39) as a matrix

$$A = \frac{1}{3}\begin{pmatrix} 1 & 1 & & \\ 1 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ & & 1 & 1 \end{pmatrix}. \tag{5.57}$$

and boundary correction vector

$$\mathbf{b}_A = \begin{pmatrix} \frac{u_l}{3} & 0 & \ldots & 0 & \frac{u_r}{3} \end{pmatrix}^T. \tag{5.58}$$

5.3 Eulerian Moving Mesh Burgers' Equation

5.3.1 Mesh Movement

We now discuss the mesh movement when solving PDEs with an Eulerian frame of reference. This movement can either be in the physical domain via static rezoning, where we must remesh after a number of time steps (can be after n time steps or when some remeshing criteria is met), or in the computational domain, where we transform our PDE using the maps from Chapter 4.

Static Rezoning

In this method, we adopt a two stage process, solving the PDE on a fixed non-uniform mesh, then remeshing. The remeshing can be performed as frequently as required.

This process is visualised in Figure 5-1, where the mesh is moved each time step. After this remeshing, the solution needs to be interpolated onto the new mesh. With this

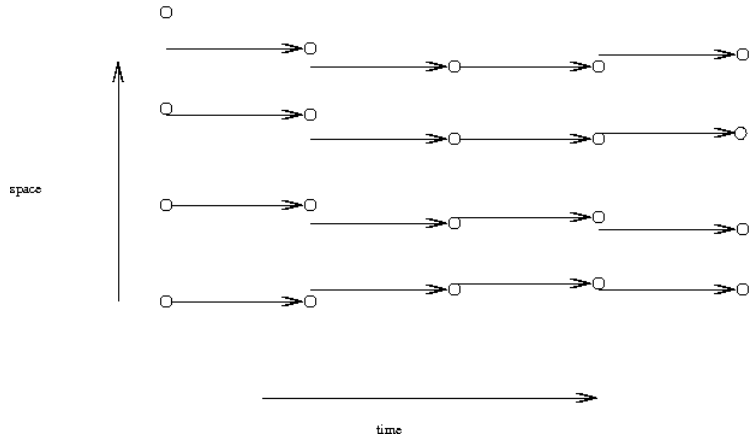

Figure 5-1: Example of using a different mesh at different times; the solution must be interpolated onto the new mesh.

two-stage strategy, we can solve the PDEs with any method, electing here to use finite difference in space and a θ-method in time.

Moving Meshes and Maps

Another approach to having a changing mesh in time, is to consider the time dependent map from a computational domain Ω_c. We can use this map with the evolving solution as a continuous problem, which itself has to be discretised, solving the PDE and moving the mesh simultaneously. This has been visualised in Figure 5-2. We consider a time dependent map

$$x = x(\xi, t)\,. \tag{5.59}$$

We now use this map as a transformation of variables, and represent our PDE in terms of the computational variable ξ, as opposed to the physical variable x.

With this transformation, we can use the chain rule to find expressions for the derivative of u with respect to x and t in terms of ξ and $x(\xi)$, as

$$u_t(\xi, t) = u_t(x, t) - x_t u_x\,, \tag{5.60}$$

$$u_\xi(\xi, t) = \frac{u_\xi(\xi, t)}{x_\xi}\,. \tag{5.61}$$

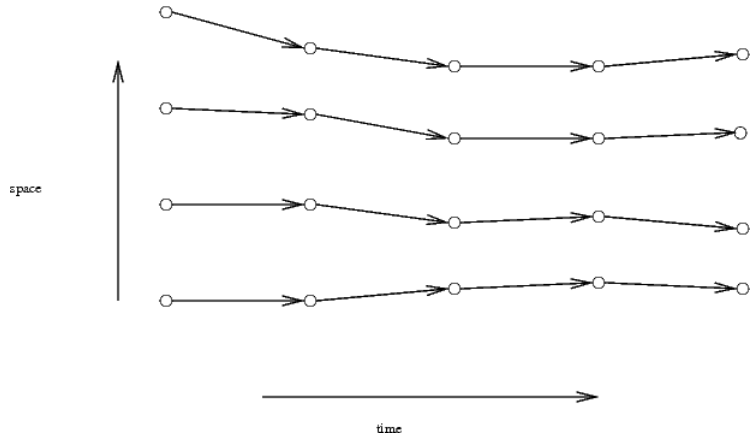

Figure 5-2: Example of mesh points in a moving mesh solution shown in the physical domain: the PDE is solved in the computational domain.

Now we can express Burgers' equation as a PDE in the (ξ, t) domain

$$u_t - \frac{u_\xi}{x_\xi}(x_t - u) - \frac{\varepsilon}{x_\xi}\left(\frac{u_\xi}{x_\xi}\right)_\xi = 0\,, \tag{5.62}$$

which we discretise (5.62) via finite differences as

$$\mathbf{U}_t = \operatorname{diag}(\delta_{2\xi}\mathbf{X})^{-1}\delta_{2\xi}\mathbf{U}(\mathbf{X}_t - A\mathbf{U}) + \varepsilon \operatorname{diag}(\delta_{2\xi}\mathbf{X})^{-1}\delta_\xi^-\left(\operatorname{diag}(\delta_\xi^+\mathbf{X})^{-1}\delta_\xi^+\mathbf{U}\right)\,, \tag{5.63}$$

where $\operatorname{diag}(\mathbf{X})$ is a matrix with the entries of $\mathbf{X}$ along the diagonal, and δ_ξ^+, δ_ξ^- and $\delta_{2\xi}$ are matrices associated with the finite difference operators (5.8–5.10). The boundary condition terms have been omitted for succinctness. To implement this mesh movement, we must discretise the mesh locations as

$$X_i(t) \approx x(\xi_i, t)\,, \tag{5.64}$$

which are then part of the solution. The transformed PDE can be coupled with a strategy for evolving X_i in time, such as the MMPDE procedure as described in Section 4.1.4, solving Burgers' equation in a moving mesh framework.

For the moving mesh we adopt the monitor function

$$\rho(x,t) = \sqrt{b + (u_x)^2}\,, \tag{5.65}$$

where $b > 0$ is a parameter which could be used to give more or less importance to very sharp gradients (note $b = 1$ gives $\rho(x,t)$ as the arc-length of $u(x,t)$). The monitor function (5.65) is smoothed and averaged, as discussed in Chapter 4 then used to evolve

the mesh $x(t)$ in time with MMPDE5

$$x_t = \frac{1}{\tau}\left(Mx_\xi\right)_\xi \,, \tag{5.66}$$

discretised as

$$(\mathbf{X})_t = \frac{1}{\tau}\delta_\xi^- \left(\sqrt{\left(b + \left(\left(\delta_\xi^+ \mathbf{X} \right)^{-1} \delta_\xi^+ \mathbf{U} \right)^2 \right)} \delta_\xi^+ \mathbf{X} \right) \,, \tag{5.67}$$

where δ_ξ^+ and δ_ξ^- are the forwards and backwards finite difference operators respectively (as described in Section 5.2), with relaxation parameter $\tau = 0.1$ and $b = 1$.

The two systems of ODEs (5.63) and (5.67) now form a system of $2N$ ODEs to be solved.

5.3.2 Temporal Discretisation of Eulerian Moving Mesh Methods

θ-method

For the static rezoning approach, we are solving Burgers' equation on a fixed mesh. We elect to use the θ-method, since it most closely mirrors the 2TL scheme employed in Chapter 3. We represent the spatial derivatives via finite difference, with the FD2C discretisation from Section 5.2 and the matrix and vector representations from Subsection 5.2.3 so that

$$\begin{aligned} \mathbf{U}_t &= -\operatorname{diag}(A\mathbf{U} + \mathbf{b}_A)(\delta_{2x}\mathbf{U} + \mathbf{b}_{\delta_{2x}}) + \varepsilon(\delta_x^2\mathbf{U} + \mathbf{b}_{\delta_x^2}) \\ &:= G(\mathbf{U})\,. \end{aligned} \tag{5.68}$$

We discretise at times t^n with

$$t^{n+1} = t^n + \Delta t\,, \tag{5.69}$$

and adopt the notation

$$\mathbf{U}^n = \mathbf{U}(t^n)\,. \tag{5.70}$$

The θ-method [33, §2.10] is

$$\frac{\mathbf{U}^{n+1} - \mathbf{U}^n}{\Delta t} = \theta_u G(\mathbf{U}^{n+1}) + (1-\theta_u)G(\mathbf{U}^n)\,, \tag{5.71}$$

which we rewrite as

$$\mathbf{U}^{n+1} - \Delta t \theta_u G(\mathbf{U}^{n+1}) = \mathbf{R}^n \,, \tag{5.72}$$

where

$$\mathbf{R}^n := \mathbf{U}^n + (1-\theta_u)\Delta t G(\mathbf{U}^n) \,. \tag{5.73}$$

We have an off-centring parameter $\theta_u \in [0,1]$ (as in the 2TL SL discretisation (2.19)), which gives the explicit Euler method for $\theta_u = 0$, backwards Euler method for $\theta_u = 1$ and the Crank-Nicholson method for $\theta_u = \frac{1}{2}$.

Equation (5.72) is a nonlinear implicit equation to be solved for $\mathbf{U}^{n+1}$. We can express this equation as a root finding problem,

$$\begin{aligned} F(\mathbf{U}^{n+1}) &:= \mathbf{U}^{n+1} - \Delta t \theta_u G(\mathbf{U}^{n+1}) - \mathbf{R}^n \,, \\ &= 0 \,, \end{aligned} \tag{5.74}$$

with tridiagonal Jacobian

$$J_F(\mathbf{U}^{n+1}) = I - \Delta t \theta_u \left(\delta_x^2 - \mathrm{diag}(A\mathbf{U} + \mathbf{b}_A)\delta_{2x} - \mathrm{diag}(\delta_{2x}\mathbf{U} + \mathbf{b}_{\delta_{2x}})A \right) \,. \tag{5.75}$$

In `burgers.m`, which can be found in Appendix A.3, Newton's method is employed to solve equation (5.74) to a specified tolerance (default of 10^-4).

ODE45

When choosing a numerical method and time step for Burgers' equation coupled with a moving mesh PDE, we must account for the stability of both discretised equations. Instead of the θ-method, we use MATLAB's inbuilt ODE solvers ODE45, which has adaptive time stepping.

The coupled system, equations (5.63) and (5.67), is solved in this manner in `fd_adaptive.m` in Appendix A.4.

5.3.3 Numerical Solutions to Eulerian Moving Mesh Burgers' Equation

We study the problem of solving the viscous Burgers' equation as an initial value problem (IVP),

$$u_t + uu_x = \varepsilon u_{xx} \quad \text{on } x \in [0,1]\,,\ t > 0 \tag{5.76}$$

with initial and boundary data

$$u(x,0) = \sin(2\pi x) + \frac{1}{2}\sin(\pi x)\,, \tag{5.77}$$

$$u(0,t) = u(1,t) = 0\,, \tag{5.78}$$

as seen in [21]. The solution develops a sharp front with width of order ε, taken here to be $\varepsilon = 0.002$, from around time $t \approx 0.2s$. This front then propagates in the positive x direction.

We discretise these equations on N moving points $X_i(t)$ and look for a numerical solution U_i, as

$$U_i(t) \approx u(X_i(t), t)\,. \tag{5.79}$$

As described above, we are using the FD2C discretisation of the nonlinear advection term from (5.39) , which is

$$(uu_x)(X_i,t) \approx \frac{(U_{i+1} + U_i + U_{i-1})\,(U_{i+1} - U_{i-1})}{3\,(X_{i+1} - X_{i-1})}\,, \tag{5.80}$$

and the standard second order finite difference approximation to the second derivative term.

θ-method

For the static rezoning approach, we take $N_t = 40$ for $t \in [0,1]$. We see the results for $N = 16$ on a static mesh and a moving mesh in Figure 5-3. We see an improvement in the results on a moving mesh, though there are small oscillations just behind the front.

We also show the numerical solution to the travelling wave problem from Chapter 3, with $\varepsilon = 0.0001$ and $\theta_u = 1/2$ in Figure 5-4 and $\theta_u = 0.7$, (typical sample for a range of experiments with $\theta_u > 0.5$) in Figure 5-5. We take $N = 64$ and $N_t = 40$, and observe the results at $t = 1.5$. We see oscillations in both cases, but with $\theta_u = 0.7$ giving a much smoother numerical solution.

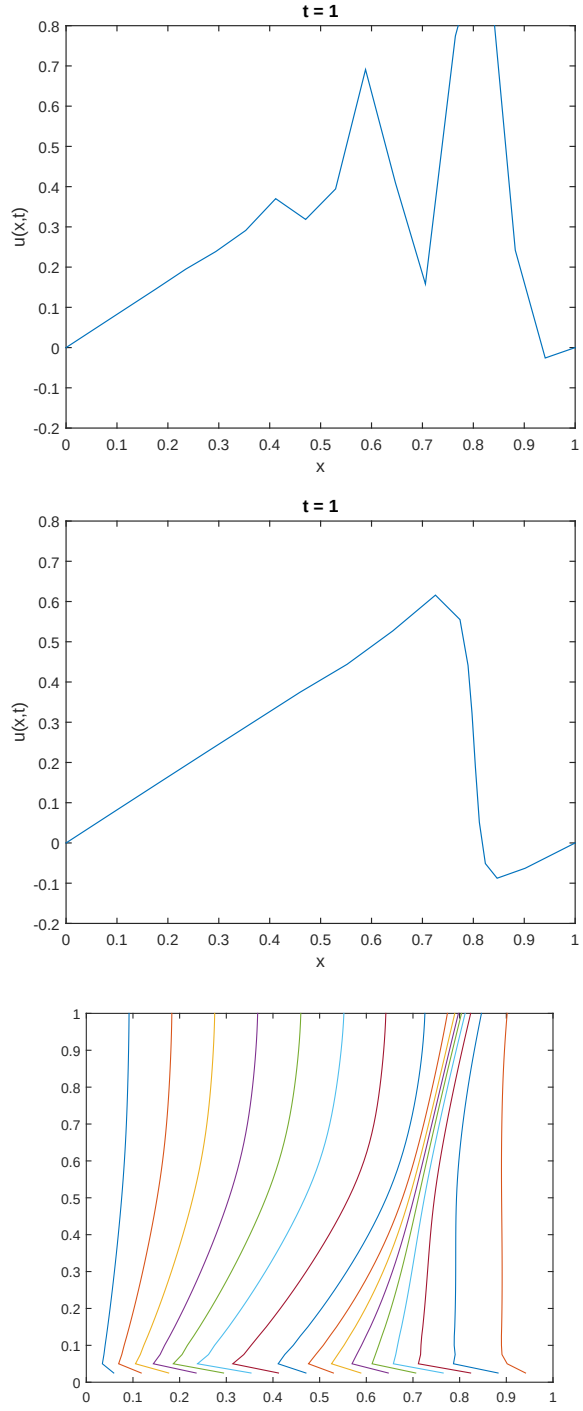

Figure 5-3: Solution to the IVP (5.76) with a θ-method for a static mesh (top) and using static rezoning with linear interpolation (middle), and the corresponding positions of the mesh points (bottom). Here $N_t = 40$, $N = 16$ and $\theta_u = 0.7$. We see oscillations in both cases, but much smaller with a moving mesh. The moving mesh has advanced slower than the reference solution (see Figure 5-6).

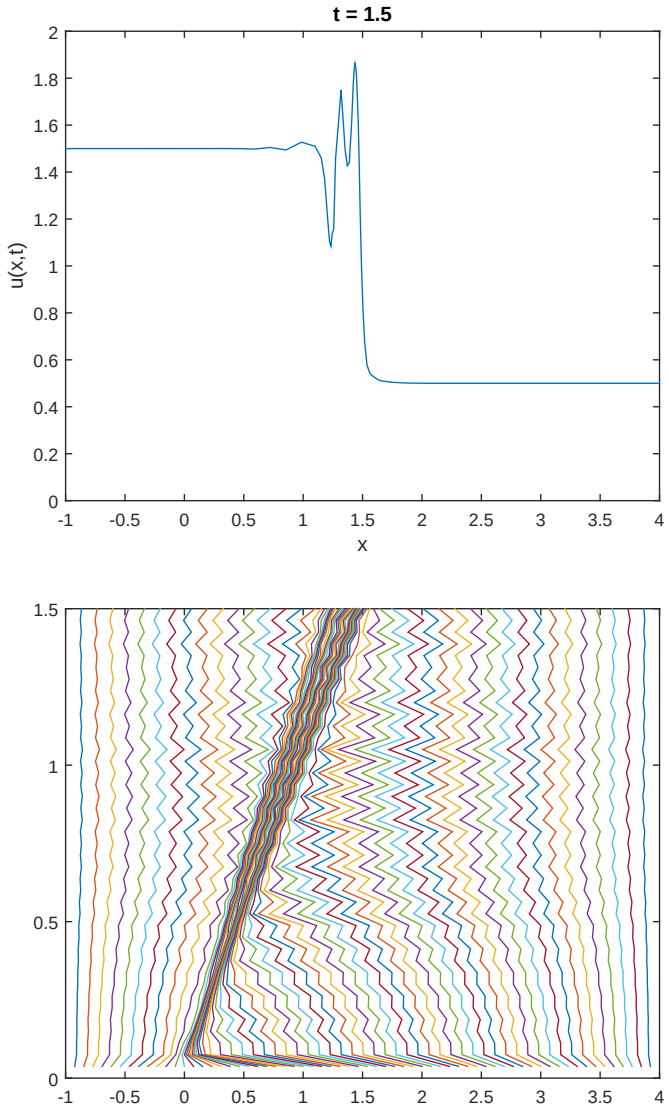

Figure 5-4: Solution to the travelling wave problem from Chapter 3 with a θ-method with static rezoning and linear interpolation. Solution at $t = 1.5$ is shown above, with the mesh movement below. Here $N_t = 40$, $N = 64$ and $\theta_u = 0.5$, giving a Crank-Nicholson scheme. We observe overshoots following the front. We also have rapid oscillations in the mesh point locations which could be dampened with some time relaxation of the monitor function.

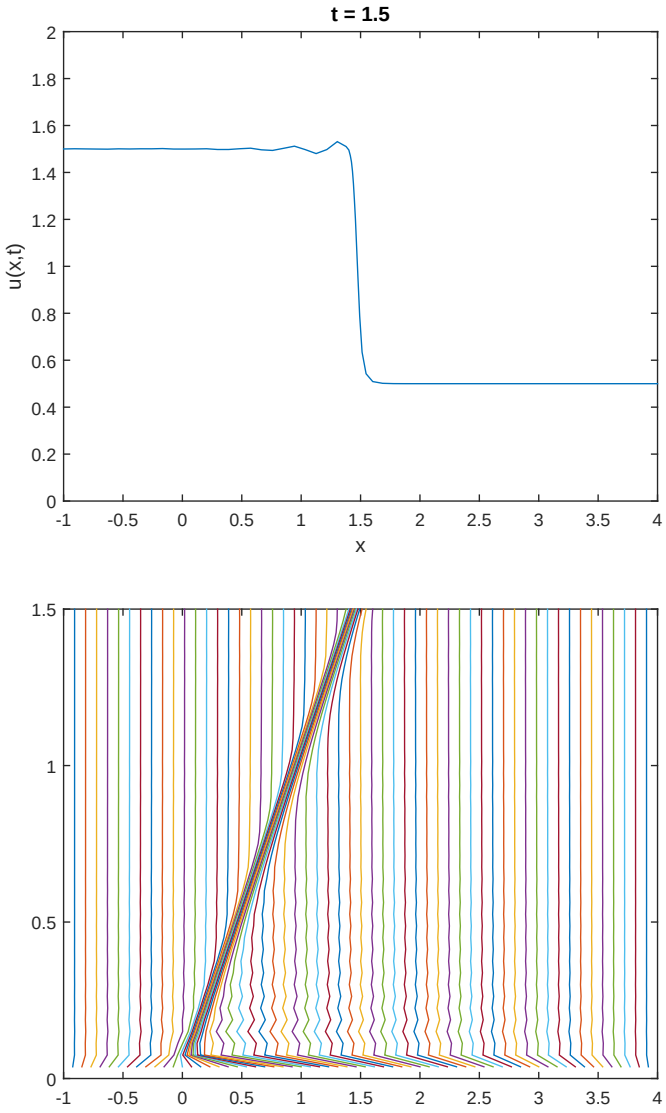

Figure 5-5: Solution to the travelling wave problem as in Figure 5-4 but with $\theta_x = 0.7$. The oscillations are still present, though much less pronounced, and the mesh movement is considerably smoother.

ODE45

We now take a closer look at the coupled Burgers'-MMPDE system.

For the coupled mesh system, resulting systems of ODEs are integrated with the MATLAB implementation of Runge-Kutta code ODE45 with standard relative tolerance of 10^{-3}.

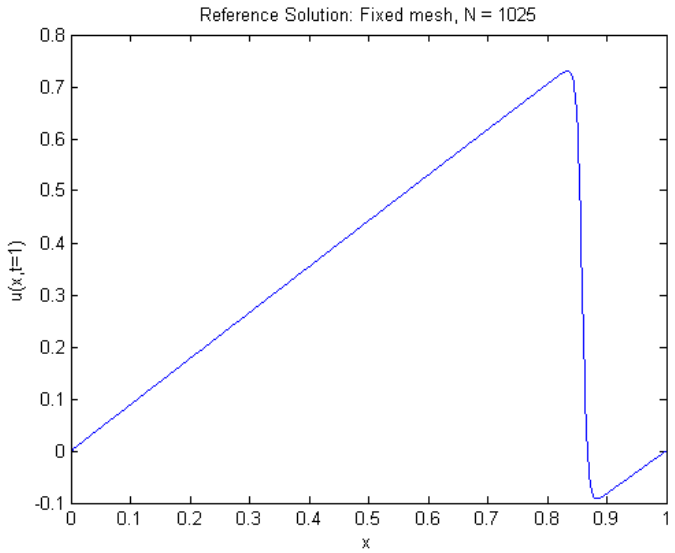

Figure 5-6: Reference solution $\mathbf{U}_{\text{ref}}$ to Burgers' equation at time $t = 1s$.

We create a reference solution $\mathbf{U}_{\text{ref}}$ on a fixed uniform mesh with $N = 1025$. Solutions with $N = 2^k + 1$, $k = \{4, \ldots, 9\}$ for $\mathbf{U}$ and X and for both uniform-static and adaptive-moving meshes are compared against $\mathbf{U}_{\text{ref}}$ at $t = 1s$, by which point the front has fully formed.

In Figure 5-6 we present the reference solution $\mathbf{U}_{\text{ref}}$, and in Figure 5-7 fixed mesh and moving mesh solutions with $N = 17$ and $N = 65$, all at $t = 1s$.

Comparing the solutions with $\mathbf{U}_{\text{ref}}$ we see the fixed mesh solution with $N = 17$ has not captured the front, while $N = 65$ has captured the front but has oscillations. With the moving mesh, both solutions appear qualitatively correct, even for the low discretisation of $N = 17$.

We can quantify these differences for each N by introducing an error vector with

$$(E_N)_i = |U_i(t = 1) - U_{\text{ref}}(X_i(t = 1), t = 1)|\,,$$

where, for the moving mesh, the value of $U_{\text{ref}}(X_i(t), t)$ has been calculated by linear interpolation from $\mathbf{U}_{\text{ref}}$ onto the moving mesh. We can then define two errors: a L_2

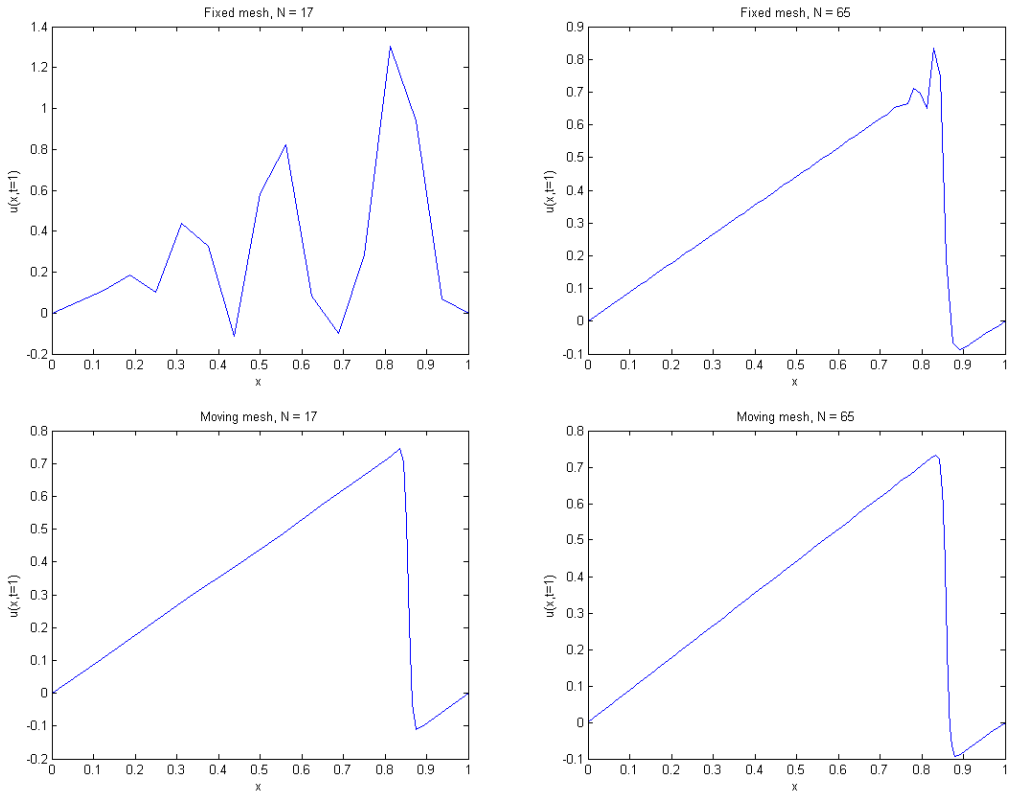

Figure 5-7: MMPDE numerical solutions to Burgers' equation at time $t = 1s$ for $N = 17$ (left) and $N = 65$ (right), on a fixed mesh (above) and moving mesh (below).

error ($||E_N||_2$) and the maximum error ($||E_N||_\infty$) as

$$||E_N||_2 = \left(\sum_{i=1}^{N} {(E_N)_i}^2 \right)^{1/2} , \tag{5.81}$$

$$||E_N||_\infty = \max_i (E_N)_i \, . \tag{5.82}$$

In Tables 5.1 and 5.2 we present these error measures for different N, along with the computation time and the minimum grid spacing on each mesh, and in Figure 5-8 we show $||E_N||_\infty$ as a function of N in each case.

Table 5.1: Numerical errors for Burgers' equation with different number of points on fixed static meshes.

N_{fixed}	$\|\|E_N\|\|_2$	$\|\|E_N\|\|_\infty$	Δx	time taken (s)
17	1.6354	1.0193	6.25×10^{-2}	0.065
33	0.4121	0.2404	3.1×10^{-2}	0.073
65	0.1647	0.1069	1.6×10^{-2}	0.014
129	0.0614	0.0422	7.8×10^{-3}	0.39
257	0.0202	0.0120	3.9×10^{-3}	1.7
513	0.0057	0.0023	2.0×10^{-3}	25
1025	-	-	9.8×10^{-4}	110

Table 5.2: Numerical errors for Burgers' equation with different number of points on an adaptive moving meshes.

N_{adaptive}	$\|\|E_N\|\|_2$	$\|\|E_N\|\|_\infty$	$\min(\Delta x_i)$	time taken (s)
17	0.1690	0.1228	6.7×10^{-3}	1.9
33	0.09	0.0370	1.7×10^{-3}	5.8
65	0.03	0.0091	7.4×10^{-4}	22
129	0.0045	0.0017	3.6×10^{-4}	79
257	0.0213	0.0031	1.8×10^{-4}	1500

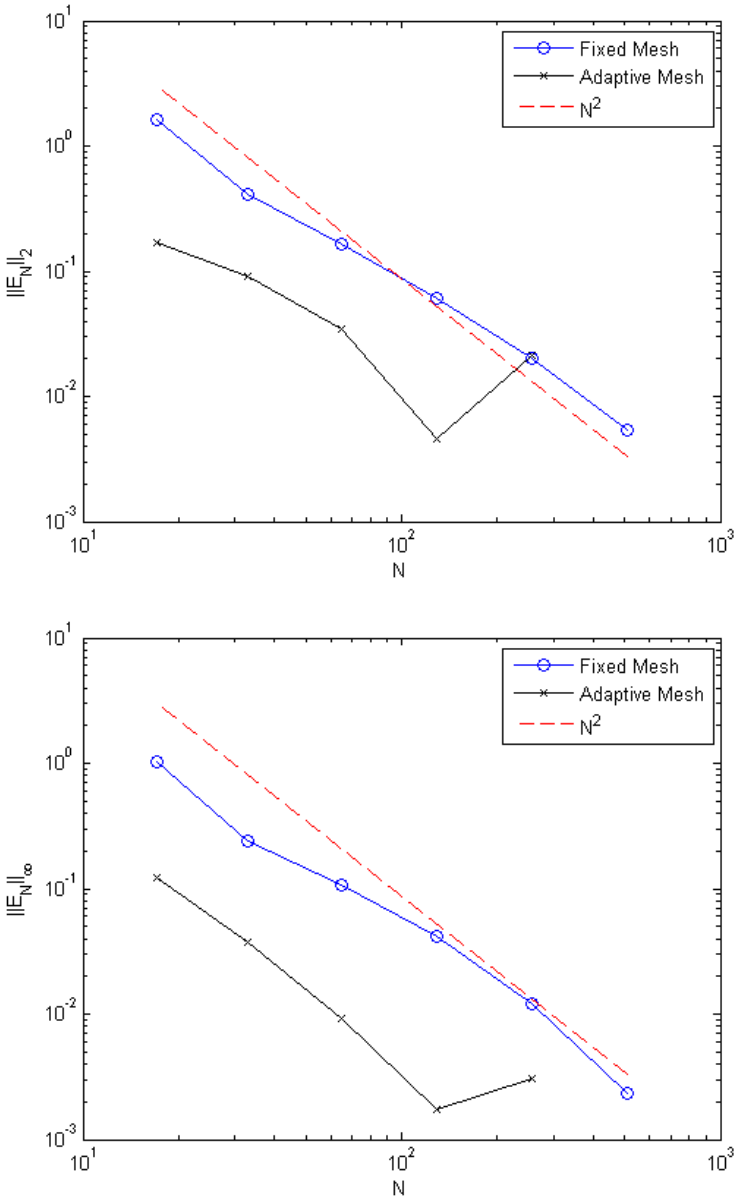

Figure 5-8: 2-norm and maximum norm of the error vector against N for the fixed and adaptive meshes. With the final point in the adaptive mesh, the reference solution is not accurate enough to compare with the numerical solution. Also shown is N^{-2}, the theoretical convergence from Section 5.2.1.

From Tables 5.1 and 5.2, and Figure 5-8, we see that we have a greatly improved solution with a moving mesh over the static mesh. For large N we require approximately $1/4$ of the grid points for a similar accuracy. For low N, the adaptive solution getting a qualitatively correct solution where the static uniform mesh fails.

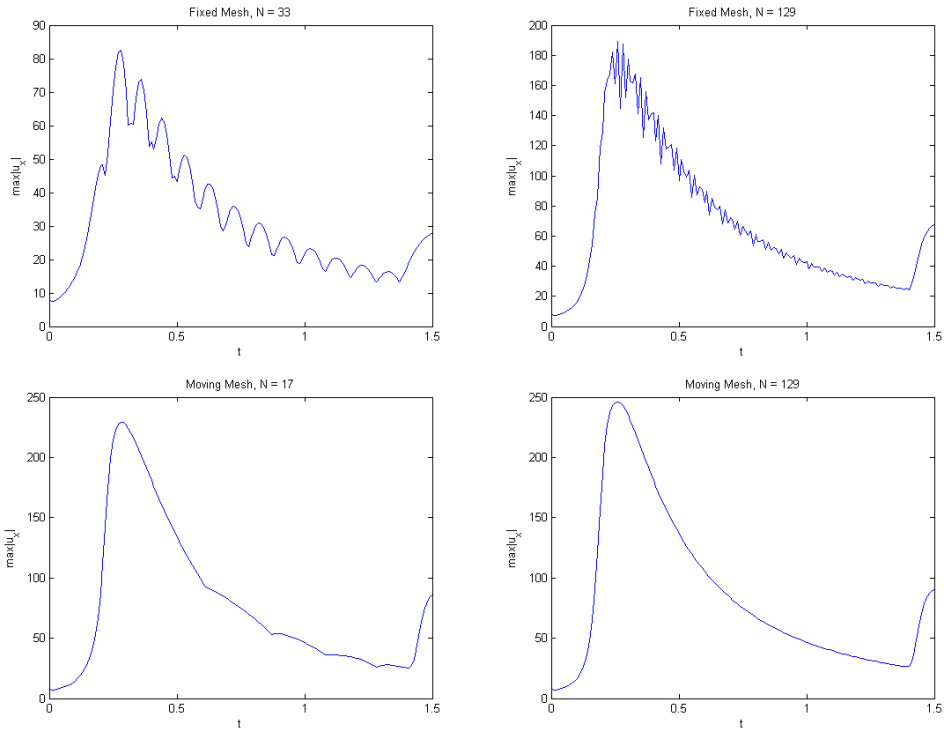

Figure 5-9: $\max |u_x|$ for different discretisations of Burgers' equation. (Above) Fixed mesh $N = 33$ and $N = 129$. (Below) Adaptive mesh $N = 17$ and $N = 129$. u_x is calculated with the forward difference operator. Note the scale of the y-axis for the fixed meshes.

In Figure 5-9 we present the maximum absolute gradient of the numerical solution (calculated here via a forward difference operator) as a function of time. We see issues with resolving with sharp gradient present at the front for low N on the fixed mesh and what appear to be oscillations even for large N. In contrast, the moving mesh solutions capture this maximum well for both low and high discretisations.

This calculation, along with that of the location of this maximum, will be useful when comparing these simultaneous solutions with that of the semi-Lagrangian discretisation, where we will see that the current implementation seems to get the estimated maximum size of u_x correct but not location and speed of the front.

5.4 Moving Mesh Semi-Lagrangian Burgers' Equation

5.4.1 Section Overview

In Chapter 3 we observed that the effect of the various errors given by the SL algorithm are both to spread the width of the front and to modify the speed of the front. The conclusion from Section 3.6 is that these errors are in general quite large, and do not vary much as the mesh size is refined. Indeed they can only be significantly reduced by taking a step size rather smaller than ε. This motivates the combination of the SL method with moving meshes with a semi-Lagrangian moving mesh method (SLMM). One way of placing spatial mesh points considered in Section 4.1.2 is to place them so that we minimise the interpolation error when approximating a function. This approach is particularly attractive in the context of a SL calculation, as we have seen in Section 3.4.2 that a principal cause of the error in this case is the process of interpolating the function defined on the mesh onto the departure points.

In this section we couple the 1D moving mesh methods with the SL discretisation, and demonstrate an improvement over the fixed uniform mesh SL calculations in Chapter 3.

5.4.2 Reminder on semi-Lagrangian discretisations

We now give a quick summary of the semi-Lagrange methods discussed in Chapter 2 and applied to Burgers' equation in Chapter 3.

We consider some quantity $G(x,t)$ with Lagrangian derivative

$$\frac{\mathrm{D}G(x,t)}{\mathrm{D}t} := \frac{\partial G}{\partial t} + \nabla G \cdot \frac{\partial x}{\partial t}\,, \tag{5.83}$$

then we can think of a Lagrangian method as a time discretisation along the pathlines of the flow. With a time discretisation of $t^{n+1} = t^n + \Delta t$ and G discretised as

$$G_i^n \approx G(X_i(t^n), t^n)\,, \tag{5.84}$$

then

$$\frac{\mathrm{D}G_i^n}{\mathrm{D}t} \approx \frac{G_i^{n+1} - G_i^n}{\Delta t}\,. \tag{5.85}$$

This is in contrast to an Eulerian discretisation, where the mesh points are fixed for the time step (in the physical or computational domain).

A Lagrangian method can be thought of as a moving mesh method where the mesh

moves with the flow, or

$$(X_i)_t = u(X_i(t), t)\,. \tag{5.86}$$

Looking at the moving frame of reference transformed Burgers' equation (5.62), the nonlinear terms cancel out, resulting in the SL Burgers' equation expressed with $x(\xi)$ as a map as in Section 5.3,

$$u_t(\xi, t) = \frac{\varepsilon}{x_\xi}\left(\frac{u_\xi}{x_\xi}\right)_\xi\,, \tag{5.87}$$

or with a Lagrangian derivative as

$$\frac{\mathrm{D}u}{\mathrm{D}t} = u_{xx}\,. \tag{5.88}$$

Burgers' equation was discretised in Chapter 3 as

$$\mathbf{U}^{n+1} - \varepsilon\Delta t\theta_u\delta^2\mathbf{U}^{n+1} = [\mathbf{U}^n + \varepsilon\Delta t(1-\theta_u)\delta^2\mathbf{U}^{n+1}]_D\,, \tag{5.89}$$

$$\mathbf{X}^{n+1} - \mathbf{X}_D^n = \frac{\Delta t}{2}(\mathbf{U}^{n+1} + \mathbf{U}_D^n)\,. \tag{5.90}$$

Fully Lagrangian methods have the disadvantage that in general the meshes can tangle in 2D and 3D flows with high vorticity, and are not widely used in practice.

A SL method uses a Lagrangian discretisation at each time step, but then interpolates onto a prescribed mesh. This means that each time level t^n has two meshes, an arrival mesh $\mathbf{X}_A$ from the previous time step and a departure mesh $\mathbf{X}_D$ going forward to the next time step. If we are using a SL method, then since we are resigned to interpolating each time step, we can use static rezoning between time steps, incurring only the cost of the mesh calculation. This is the approach used in the remainder of this chapter.

5.4.3 Moving Mesh Calculations

We now discuss the mesh movement, as with static rezoning.

We are looking to represent $u(x)$ on N mesh points, where the mesh points are free to move during the course of the calculation. As with the static rezoning, $\rho(x)$ is a monitor function which is a measure of the error of the resulting approximation. We want to equidistribute the mesh points at specific times, such that

$$\int_{X_j}^{X_{j+1}} \rho(x)\mathrm{d}x = \frac{\theta}{N}\,. \tag{5.91}$$

As shown in Chapter 3 the interpolation error makes a significant contribution to the

overall error in a SL method. When using a linear interpolant to evaluate $u(x)$, we can minimise the interpolation error (2.35) by taking a curvature mesh density function (4.19), as

$$\rho(x) = \sqrt{a^2 + b^2 u_{xx}^2}\,. \tag{5.92}$$

The constants a and b are chosen carefully for normalisation so that the integral of ρ over the domain is 1 [5, §2.8.2]. The mesh points X_j can then be calculated from the expression (5.91) by quadrature, described in Section 4.1.3. We compare the use of this interpolation error-minimising curvature monitor function with the commonly used arc-length monitor function (4.18) given by

$$\rho(x) = \sqrt{a^2 + b^2 u_x^2}\,. \tag{5.93}$$

In the context of the SL calculation, the function $u(x,t)$, and hence the monitor function, is not known a-priori but is calculated as part of the solution. However, we may implement a two-stage strategy to calculate a moving mesh X_j^n at each time level t^n which makes use of the expression (5.91). We note that in many moving mesh strategies (see for example [50]) it is required to interpolate the solution onto the new mesh once it has been calculated. As discussed above, this step is not required in the algorithm we propose, as the interpolation onto the departure points automatically deals with this issue.

The moving mesh algorithm coupled to SL

The algorithm for coupling a moving mesh method to the SL algorithm augments the SL algorithm to the MMSL algorithm in a two-step process as follows.

1. At time level t^n we have a computed solution $\mathbf{U}^n$ and a computed arrival mesh $\mathbf{X}^n$ (at time level t^0 the solution $\mathbf{U}^0$ is the value of the initial state and the mesh $\mathbf{X}^0$ is assumed to be uniform).

2. Using the known values of $\mathbf{U}^n$ a new arrival mesh $\mathbf{X}^{n+1}$ is calculated to equidistribute the monitor function $\rho(x)$ evaluated on $\mathbf{U}^n$.

3. Using the new arrival mesh $\mathbf{X}^{n+1}$, a new solution $\mathbf{U}^{n+1}$ is calculated at these points using exactly the same SL algorithm as described in Chapter 3.

Note that this procedure is essentially constructing a mesh at time t^{n+1} which equidistributes the monitor function of the solution at time t^n. This means the mesh

is "lagged" behind the solution by one time step, but for a small enough time step this should not be an issue. This lag could be reduced by calculating a "predictor" solution $\tilde{\mathbf{U}}^{n+1}$ for t^{n+1}, equidistributing the mesh with respect to $\tilde{\mathbf{U}}^{n+1}$, then calculating a "corrector" solution $\mathbf{U}^{n+1}$.

The main new part of this computation is the calculation of the new arrival points X^{n+1} in Step 2. This is done as follows:

1. Using the values of $\mathbf{U}^n$ on the current arrival points $\mathbf{X}^n$ the monitor function ρ^n is evaluated at each point $\mathbf{X}^n$ from (5.92) using a finite difference approximation as in Section 5.2.

2. The resulting values of ρ^n are then *smoothed* using a low pass filter from Section 4.2.4 to give points $\hat{\rho}^n$. This has proved necessary in earlier calculations of moving meshes (see for example [34]) to ensure a smooth and regular mesh at each stage of the calculation.

3. A linear interpolant is used to reconstruct a continuous monitor function $\rho(x)$ from the smoothed points $\hat{\rho}^n$.

4. The values of the new arrival points $\mathbf{X}^{n+1}$ are then calculated from the equidistribution equation (5.91). In this calculation, the linear interpolating function is integrated exactly to give a piecewise quadratic function. The $\mathbf{X}^{n+1}$ are then found by solving a locally quadratic equation, implemented by the MATLAB code `equidistribute.m` in Appendix A.2.

The above two algorithms can be seen in Figure 5-10, the first being with $\mathcal{L} = 1$.

The use of the moving mesh means that the underlying function is much better represented locally, with the mesh spacing being of the order of the shock width ε. Thus the effective size of the mesh Δx is much smaller in the equations for the numerical viscosity (3.82) and the numerical front speed (3.72), leading to smaller errors for both.

5.4.4 Results for Smooth Initial Conditions

We solve the SL Burgers' equations with $\varepsilon = 0.002$ and initial and boundary conditions

$$u(x,0) = \sin(2\pi x) + \frac{1}{2}\sin(\pi x)\,, \tag{5.94}$$

$$u(0,t) = u(1,t) = 0,. \tag{5.95}$$

1: $\mathbf{X}^0$ and $\mathbf{U}^0$ given
2: **for** $n := 0$ **to** $N_t - 1$ (*Time loop*) **do**
3: Make in initial estimate to $\mathbf{U}^{n+1\{0\}}$ and $\mathbf{X}^{n+1\{0\}}$
4: **for** $\ell := 1$ **to** $\mathcal{L}$ (*Mesh loop*) **do**
5: Calculate $\rho(x)$ from $\mathbf{U}^{n+1\{\ell-1\}}$
6: Find $\mathbf{X}^{n+1\{\ell\}}$ which equidistributes $\rho(x)$ via (5.91)
7: Calculate $\mathbf{U}^{n+1\{\ell\}}$ with the algorithm described in Figure 3-1 but with the non-uniform mesh $\mathbf{X}^{n+1\{\ell\}}$
8: **end for**
9: **end for**

Figure 5-10: Solution procedure for a moving mesh semi-Lagrangian method. If $\mathcal{L} = 1$ then the mesh is delayed by one time step. For step 7, if $\ell < \mathcal{L}$ we may want to use a cheaper, less accurate method to calculate $\mathbf{U}^{n+1\{\ell\}}$, which is "good enough" to use to calculate the mesh $\mathbf{X}^{n+1\{\ell\}}$.

This was solved with a version of the MATLAB code `burgers.m` from Appendix A.3.

We find $\mathbf{X}_A^{n+1}$ with trapezium equidistribution (see Section 4.1.3) using the monitor function

$$M(X_{i,A}^n, t^n) = \sqrt{1 + (\delta_x U_i^n)^2}\,, \tag{5.96}$$

with no smoothing or averaging. The value of $\mathbf{U}_D$ is evaluated with linear interpolation. As with with experiments in Section 5.3.3 the solutions are presented at $t = 1s$. We see the front and mesh movement in Figures 5-11 and 5-12.

These results show a good representation of the front shape, but the front speed is slower than expected.

We now apply the moving mesh semi-Lagrangian scheme to the travelling wave problem from Chapter 3.

5.4.5 Travelling Wave Results

We now consider an implementation of the MMSL algorithm to Burgers' equation, with boundary conditions $u(-\infty, t) = c + \alpha,\ u(\infty) = c - \alpha$ with $c = 1$ and $\alpha = 0.1$, as in Section 3.5.

In this algorithm we take a fixed number N of mesh points over the same x interval $x \in [-1, 4]$, with N increasing from 20 to 1000. We also take time steps of $N_t = 40$, 80 and 160. The mesh points are then moved to equidistribute the monitor function taken as

$$\rho(X_i) = \sqrt{0.1 + (\delta_x^+ U_i)^2}\,. \tag{5.97}$$

The monitor function is smoothed during the calculation and approximated by a

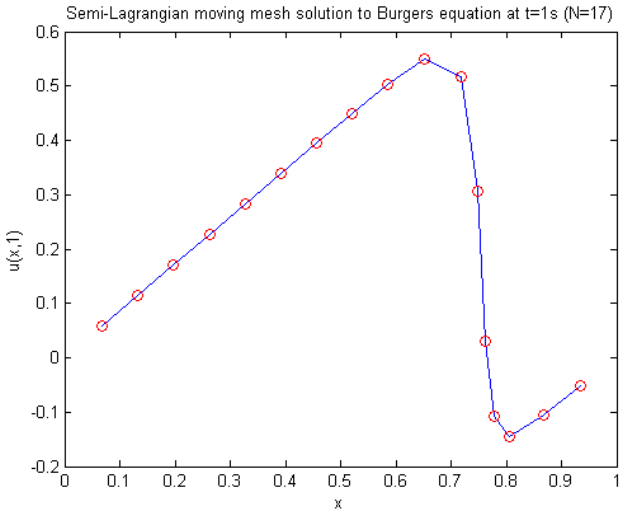

Figure 5-11: Numerical solution to Burgers equation at $t = 1s$ with a Semi-Lagrange discretisation and a moving mesh. The mesh is determined by trapezium equidistribution to the u arc-length monitor function. The front is propagating but slower than expected, ending at a point just under $x = 0.8$, whereas the reference solution in Figure 5-6 finishes beyond $x = 0.85$.

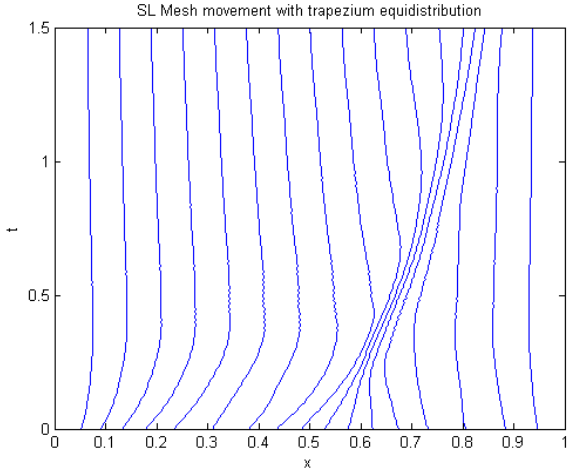

Figure 5-12: The mesh movement for the solution in Figure 5-11.

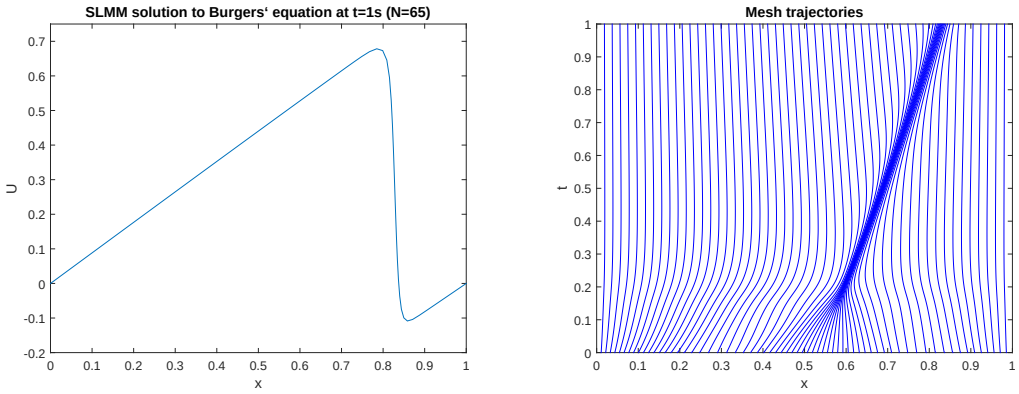

Figure 5-13: SLMM numerical solution to Burgers' equation at $t = 1s$ for $N = 65$. The travelling front has moved further than that seen in the lower resolution run in Figures 5-11 and 5-12.

piecewise-linear function which is then exactly equidistributed.

As a first experiment we take $N = 80$ and $\Delta t = 80$ and plot both the evolution of the arrival points and the minimum spacing Δ_{min} of the arrival points, seen in Figure 5-14. As seen in the expression (3.81), at the front we require a mesh width smaller than 4×10^{-3} to overcome the minimum viscosity. We see that this mesh satisfies this condition, and gives $\widehat{\varepsilon} = 5 \times 10^{-4}$ and $k = 1.02$.

As a second calculation we vary N and Δt and consider the resulting errors. The results are presented in Figure 5-15 in which we see the effective diffusion parameter, and in Figure 5-16 in which we see the front speed. We can compare these with the corresponding results in Chapter 3 (Figures 3-10 and 3-9).

In both cases we see a very significant improvement over the calculations using a fixed mesh.

In Figure 5-15 we see that for $N > 100$ the effective diffusion parameter $\widehat{\varepsilon}$ is very close indeed to the true value of $\varepsilon = 10^{-4}$. Furthermore, we do not see the oscillations in the value of the calculated diffusion parameter $\widehat{\varepsilon}$ which were observed earlier and which were due to the CFL condition. Note, however, that as in the case of the static mesh and for the two examples considered, the error measured in terms of the size of $\widehat{\varepsilon}$ *increases* as the step size Δt *decreases*. This is partially due to more interpolation steps being performed.

In Figure 5-16 we see that the calculated speed k is much closer to the true speed

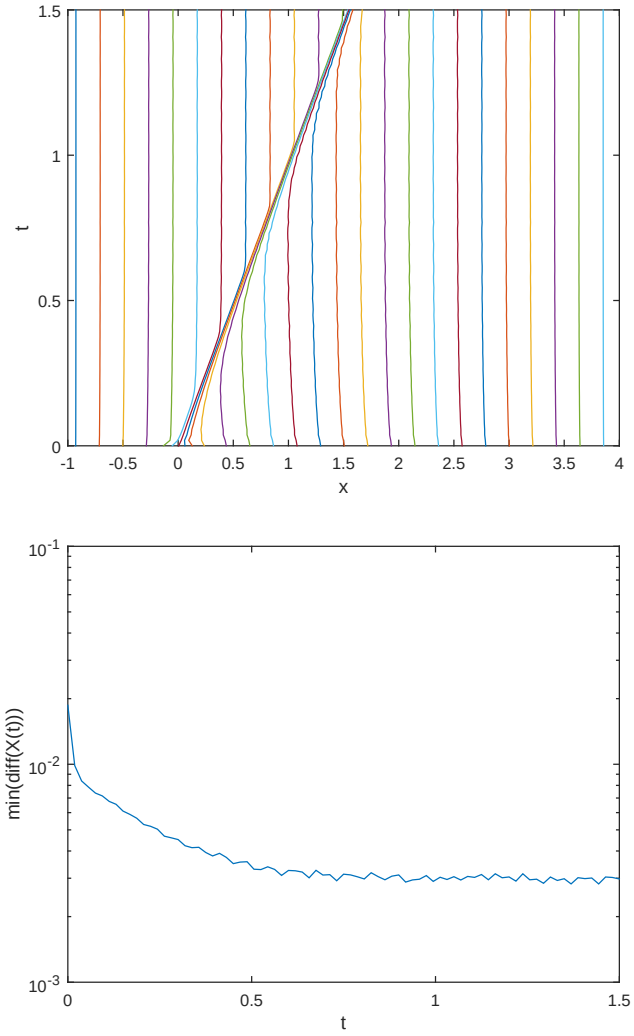

Figure 5-14: 26 mesh point trajectories (left) and the minimum mesh spacing over time (right) for a moving mesh solution to Burgers' equation.

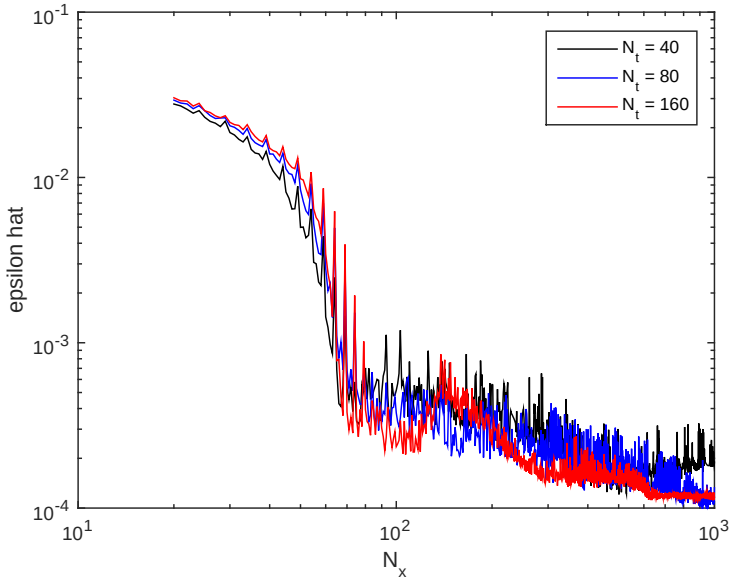

Figure 5-15: Parameter $\widehat{\varepsilon}$ with a moving mesh and curvature based monitor function.

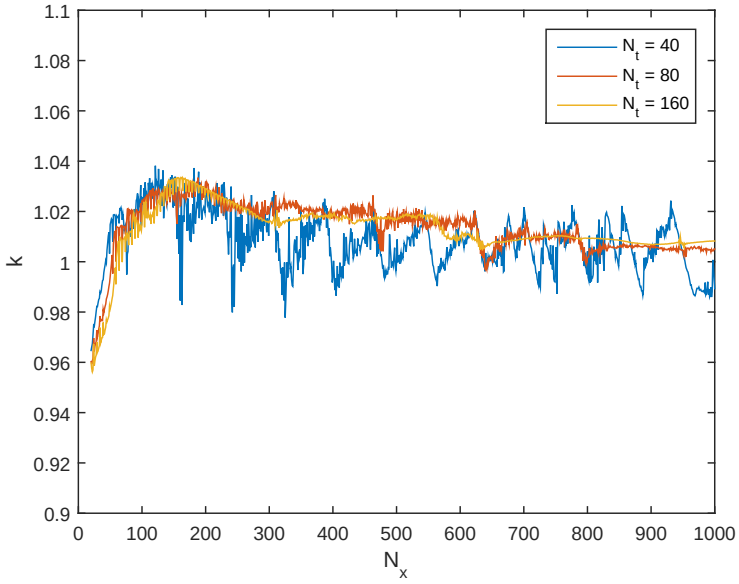

Figure 5-16: Front speed k with a moving mesh and curvature based monitor function.

of $c = 1$. Furthermore, in line with the earlier calculation, this error decreases as Δt decreases and there is no evidence of oscillation. Thus the calculated front speed is a more reliable estimate of the true speed in all cases for this example.

Chapter 6

Vertical Column Model: Fixed Non-Uniform Mesh

In this chapter we experiment with static non-uniform meshes in a NWP setting. For this, we apply the earlier theory on SISL methods and static adaptivity on a more meteorological problem. We construct an atmospheric simulation model with which to work, then compare numerical simulations with and without static adaptivity. We want our model to be simple, close in form to the relevant set of equations used in NWP and be discretised in a manner that mirrors the formulation of ENDGame, the Met Office's dynamical core currently in use [56, 32]. Accordingly, we present a non-hydrostatic vertical column model for both dry and moist air, and study the SISL method applied to this.

This chapter is laid out as follows: In Section 6.1 we show some of the derivation of the vertical column model and some useful hydrostatic profiles. In Section 6.2 we show a SL discretisation of the vertical column model similar to that used in ENDGame, and present the linearisation in Section 6.3, finite difference spatial discretisation in 6.4 and the resulting Helmholtz equation to be solved for pressure in Section 6.5. Section 6.6 documents some experiments showing the effect of discretisation error from different static meshes. And Section 6.7 shows the derivation of some of the terms used when modelling moisture in the vertical column, with the PDEs for the moisture species in Section 6.8.

6.1 Vertical Column Formulation

In this section we derive the form of the vertical column equations that forms the basis of our experiments. This will be a dry adiabatic non-hydrostatic vertical column with

no external forcing. Following this, we present a method for solving these equations that closely mirrors the approach used by the Met Office.

Our equations are derived from the following physical laws: the conservation of mass, the conservation of momentum, the conservation of energy, and the equation of state (linking the pressure, density and temperature of air). The model is based on the Met Office Dynamical Core [32, 56], with theory from [18].

The thermodynamic equation is given in [18, (2.41)] as

$$c_v \frac{\mathrm{D}T}{\mathrm{D}t} + p \frac{\mathrm{D}}{\mathrm{D}t}\left(\frac{1}{\rho}\right) = \dot{Q}\,, \tag{6.1}$$

where T is the temperature, ρ is the density, p is the pressure, c_v is the specific heat of dry air at constant volume and $\dot{Q}$ is the energy transfer.

We also use the equation of state, [18, (1.25)]

$$p = \rho R T\,, \tag{6.2}$$

where R is the specific gas constant for dry air. Differentiating the logarithm of (6.2) gives

$$\frac{1}{p}\frac{\mathrm{D}p}{\mathrm{D}t} = \frac{1}{\rho}\frac{\mathrm{D}\rho}{\mathrm{D}t} + \frac{1}{T}\frac{\mathrm{D}T}{\mathrm{D}t}\,. \tag{6.3}$$

Using the state equation with Mayer's relation [18, §2.7]

$$R = c_p - c_v\,, \tag{6.4}$$

where c_p is the specific heat of dry air at a constant pressure, we can rewrite the thermodynamic equation (6.1) as

$$c_p \frac{\mathrm{D}T}{\mathrm{D}t} - \frac{1}{\rho}\frac{\mathrm{D}p}{\mathrm{D}t} = \dot{Q}\,. \tag{6.5}$$

We introduce the derived variables Exner pressure, [18, (4.47)] defined as

$$\Pi := \left(\frac{p}{p_0}\right)^{R/c_p} \tag{6.6}$$

for a reference surface pressure p_0, and potential temperature, defined as

$$\theta := \frac{T}{\Pi}\,. \tag{6.7}$$

Now the thermodynamic equation (6.5) can be expressed in the form

$$c_p \Pi \frac{\mathrm{D}\theta}{\mathrm{D}t} = \dot{Q} \,. \tag{6.8}$$

For our adiabatic model, there is no energy transfer out of the system, hence $\dot{Q} = 0$ and equation (6.8) gives

$$\frac{\mathrm{D}\theta}{\mathrm{D}t} = 0 \,. \tag{6.9}$$

For a vertical column, we ignore any horizontal variations (changes in x- and y-directions), and the Coriolis force. The momentum equation [18, (2.21)] reduces down to

$$\frac{\mathrm{D}w}{\mathrm{D}t} + \frac{1}{\rho}\frac{\partial p}{\partial z} + g = 0 \,, \tag{6.10}$$

where w is the vertical wind. Using the variables Π and θ, this vertical momentum equation and the state equation (6.2) take the form

$$\frac{\mathrm{D}w}{\mathrm{D}t} + c_p \theta \frac{\partial \Pi}{\partial z} + g = 0 \,, \tag{6.11}$$

$$\Pi^{\frac{1-\kappa}{\kappa}} = \frac{R}{p_0}\rho\theta \,, \tag{6.12}$$

where

$$\kappa := \frac{c_p}{R} \,. \tag{6.13}$$

Finally, the conservation of mass can be expressed in our 1D setting as

$$\frac{\mathrm{D}\rho}{\mathrm{D}t} + \rho\frac{\partial w}{\partial z} = 0 \,. \tag{6.14}$$

For the boundary conditions, we prescribe the fields with zero velocity at the surface, hence

$$w(0) = 0 \,, \tag{6.15}$$

$$\Pi(0) = \Pi_{\mathrm{surf}} \,, \tag{6.16}$$

$$\theta(0) = \theta_{\mathrm{surf}} \,, \tag{6.17}$$

and $\rho(0)$ chosen to satisfy the equation of state (6.12).

At our model ceiling we impose a zero velocity condition

$$w(z_{\mathrm{top}}) = 0 \,. \tag{6.18}$$

The equations (6.9), (6.11), (6.12) and (6.14) now form the model equations with boundary conditions (6.15–6.18). In our numerical experiments we use initial conditions with $w = 0$, and ρ, θ and Π sampled from hydrostatic profiles, which we look at in the next section.

6.1.1 Hydrostatic profiles for the vertical column

It is useful to look at some static non-evolving solutions to the vertical column equations. Initially we discuss the general procedure of building hydrostatic profiles, then give some examples which are used to initialise numerical experiments in Section 6.6.

For a static profile we have $w = 0$ and we need to specify the three variables ρ, θ and Π. For this purpose, we require three equations: the equation of hydrostatic balance

$$c_p \theta \frac{\partial \Pi}{\partial z} + g = 0 \tag{6.19}$$

(the momentum equation (6.11) with $w = 0$), a constraint equation and the equation of state (6.12). In general we use hydrostatic balance and the constraint to obtain θ and Π, then the state equation to get ρ.

Some typical hydrostatic profiles with corresponding constraints are isentropic, isothermal and constant buoyancy,

$$N^2 := \frac{g}{\theta}\frac{\partial \theta}{\partial z} = \text{constant}\,. \tag{6.20}$$

We can also prescribe an initial field, such as $\theta = \theta_0(z)$ or $T = T_0(z)$.

We now give some examples of these profiles. In each case, ρ is defined by the equation of state and not given here. An *isentropic profile* has a constant potential temperature

$$\theta(z) = \theta_{\text{surf}} \quad \forall z\,, \tag{6.21}$$

with θ_{surf}, the potential temperature at the surface. For our profile to be static we then have uniformly $w = 0$, and our momentum equation (6.11) gives a linear Exner pressure

$$\Pi(z) = \Pi_{\text{surf}} - \frac{g}{c_p \theta_{\text{surf}}} z\,. \tag{6.22}$$

An *isothermal profile* has a constant temperature

$$T(z) = \Pi(z)\theta(z) = T_{\text{surf}}\,, \tag{6.23}$$

and the hydrostatic equation takes the form

$$\begin{aligned} \frac{\partial \Pi}{\partial z} &= \frac{-g\Pi}{c_p T_{\text{surf}}}\,, \\ \Pi(0) &= \Pi_{\text{surf}}\,, \end{aligned} \tag{6.24}$$

with solution

$$\Pi(z) = \Pi_{\text{surf}} \exp\left(\frac{-g}{c_p T_{\text{surf}}} z\right)\,, \tag{6.25}$$

$$\theta(z) = \theta_{\text{surf}} \exp\left(\frac{g}{c_p T_{\text{surf}}} z\right)\,. \tag{6.26}$$

$$\tag{6.27}$$

Finally we construct an idealised *inversion layer* between two heights z_{IB} (bottom of the inversion layer) and z_{IT} (top of the inversion layer). We specify a temperature beneath the layer T_{surf} and a temperature above the layer T_{top}, with the temperature varying linearly between the inversion layer boundaries z_{IB} and z_{IT}, so that

$$T(z) = \begin{cases} T_{\text{surf}} & 0 \le z < z_{IB}\,, \\ T_{\text{surf}} + T_z(z - z_{IB}) & z_{IB} \le z < z_{IT}\,, \\ T_{\text{top}} & z_{IT} \le z\,, \end{cases} \tag{6.28}$$

with

$$T_z = \frac{T_{\text{top}} - T_{\text{surf}}}{z_{IT} - z_{IB}}\,. \tag{6.29}$$

A careful calculation using the hydrostatic equation (6.19) shows that the Exner pressure and potential temperature are continuous and satisfy the surface conditions

and momentum equation (6.11) with $w = 0$ as

$$\Pi(z) = \begin{cases} \Pi_{\text{surf}} \exp\left(\frac{-g}{c_p T_{\text{surf}}} z\right) & 0 \le z < z_{IB}\,, \\ A T(z)^{-g/(c_p T_z)} & z_{IB} \le z < z_{IT}\,, \\ B \exp\left(\frac{-g}{c_p T_{\text{top}}} z\right) & z_{IT} \le z\,, \end{cases} \tag{6.30}$$

$$\theta(z) = \begin{cases} \theta_{\text{surf}} \exp\left(\frac{g}{c_p T_{\text{surf}}} z\right) & 0 \le z < z_{IB}\,, \\ A^{-1} T(z)^{(g/(c_p T_z)+1)} & z_{IB} \le z < z_{IT}\,, \\ C \exp\left(\frac{g}{c_p T_{\text{top}}} z\right) & z_{IT} \le z\,, \end{cases} \tag{6.31}$$

with constants

$$A = \Pi(z_{IB}) T_{\text{surf}}^{g/c_p T_z}\,, \tag{6.32}$$

$$B = \Pi(z_{IT}) \exp\left(\frac{g}{c_p T_{\text{top}}} z_{IT}\right)\,, \tag{6.33}$$

$$C = \theta(z_{IT}) \exp\left(\frac{-g}{c_p T_{\text{top}}} z_{IT}\right)\,. \tag{6.34}$$

In contrast to this piecewise linear temperature profile, we can use a smoother $T(z)$, for example a tanh temperature profile,

$$T(z) = T_{\text{mid}} + \frac{T_{\text{w}}}{2} \tanh\left(\frac{4(z - z_{\text{IM}})}{\epsilon}\right)\,, \tag{6.35}$$

with $T_{\text{mid}} = (T_{\text{surf}} + T_{\text{top}})/2$, $T_{\text{w}} = T_{\text{top}} - T_{\text{surf}}$, $z_{\text{IM}} = (z_{\text{IB}} + z_{\text{IT}})/2$ and $\epsilon = z_{\text{IT}} - z_{\text{IB}}$. The piecewise linear temperature and tanh-temperature profiles are shown in Figure 6-1.

The tanh temperature profile has a hydrostatic solution with

$$\Pi(z) = A \exp\left(\frac{4(z - z_{\text{IM}})}{\epsilon} T_{\text{surf}} \phi\right) \left(\left(\tanh\left(\frac{4(z - z_{\text{IM}})}{\epsilon}\right) + 1\right) T(z)^{-1}\right)^{(T_{\text{w}}\phi/2)}\,, \tag{6.36}$$

$$\theta(z) = \frac{T(z)}{\Pi(z)}\,, \tag{6.37}$$

with $\phi = \frac{-\epsilon g}{4 c_p T_{\text{surf}} T_{\text{top}}}$ and A fixed such that $\Pi(0) = \Pi_{\text{surf}}$.

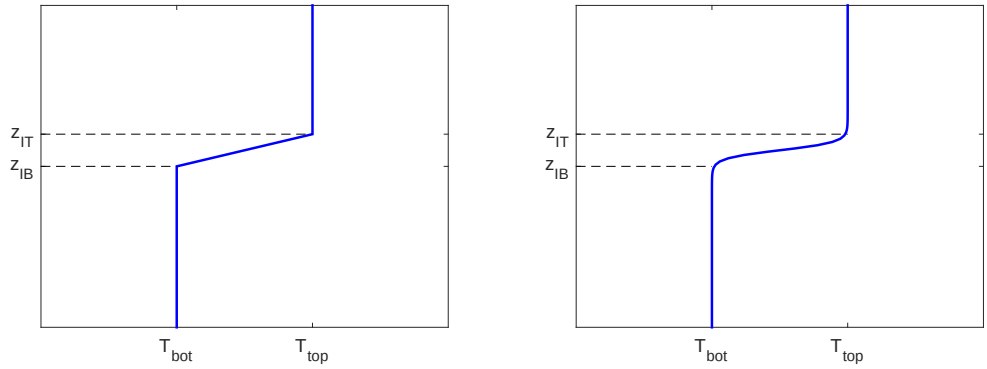

Figure 6-1: Piecewise linear temperature profile (left) from equation (6.28), and a tanh-temperature profile (right) from equation (6.35).

We now return to the model equations, and show two ways in which they can be discretised in a semi-Lagrangian manner.

6.2 Temporal Discretisation (SISL)

In this section we give a semi-Lagrangian two-time-level discretisation of the vertical column model from Section 6.1, which we repeat below. This formulation is similar to the discretisation in ENDGame [57].

The consolidated vertical column model equations are

$$\frac{\mathrm{D}w}{\mathrm{D}t} + c_p\theta\frac{\partial \Pi}{\partial z} + g = 0\,, \tag{6.38}$$

$$\frac{\mathrm{D}\rho}{\mathrm{D}t} + \rho\frac{\partial w}{\partial z} = 0\,, \tag{6.39}$$

$$\frac{\mathrm{D}\theta}{\mathrm{D}t} = 0\,, \tag{6.40}$$

over $z \in [0, z_{top}]$ subject to the state equation

$$\Pi^{\frac{1-\kappa}{\kappa}} - \frac{R}{p_0}\rho\theta = 0\,, \tag{6.41}$$

and the boundary conditions

$$w(0) = 0\,, \tag{6.42}$$

$$w(z_{\text{top}}) = 0\,, \tag{6.43}$$

$$\Pi(0) = \Pi_{\text{surf}}\,, \tag{6.44}$$

$$\theta(0) = \theta_{\text{surf}}\,. \tag{6.45}$$

This system (6.38-6.41) has been expressed with the kinematic equation, which in 1D is

$$\frac{\mathrm{D}z}{\mathrm{D}t} = w\,, \tag{6.46}$$

giving us the required form for using a SL discretisation, to be solved for w, ρ, θ and Π.

We discretise these in the two-time-level semi-Lagrangian fashion, as described in Chapter 2, which was also applied in Chapter 3 to Burgers' equation. The terms are discretised along a semi-Lagrangian trajectory. Source terms are averaged along the trajectory in a off-centred way with parameters α, β with $\alpha+\beta = 1$, or a centred Crank-Nicholson-like discretisation for $\alpha = \beta = 0.5$. The terms α and β are equivalent to θ_u and $(1-\theta_u)$ from Chapter 2. The time-discretised equations using the SL framework are hence from equations (6.38–6.41)

$$\left[w + \Delta t\alpha\left(c_p\theta\frac{\partial\Pi}{\partial z} + g\right)\right]_A^{n+1} = \left[w - \Delta t\beta\left(c_p\theta\frac{\partial\Pi}{\partial z} + g\right)\right]_D^{n}\,, \tag{6.47}$$

$$\left[\rho + \Delta t\alpha\rho\frac{\partial w}{\partial z}\right]_A^{n+1} = \left[\rho - \Delta t\beta\rho\frac{\partial w}{\partial z}\right]_D^{n}\,, \tag{6.48}$$

$$[\theta]_A^{n+1} = [\theta]_D^{n}\,, \tag{6.49}$$

$$\left[\Pi^{\frac{1-\kappa}{\kappa}} - \frac{R}{p_0}\rho\theta\right]_A^{n+1} = 0\,, \tag{6.50}$$

$$\tag{6.51}$$

with

$$z_D = z_A - \Delta t\left[\alpha_z w_A^{n+1} + (1-\alpha_z)w_D^{n}\right]\,, \tag{6.52}$$

from (6.46).

The SISL solution procedure was described in a general setting in Chapter 2 and for Burgers' equation in Chapter 3. We repeat it here in Figure 6-2 in the setting of our vertical column model.

We have a *time loop* in which the fields w, ρ, θ and Π are temporally updated, an

```
1: for n := 1 to n_max (Time Loop) do
2:    Estimate z_D (use initial guess of w(z, t^{n+1}) = w(z, t^n))
3:    for ℓ := 1 to L (Outer Loop) do
4:       Interpolate right-hand sides of equations (6.47–6.49) onto z_D
5:       for k := 1 to K (Inner Loop) do
6:          Iteratively solve (6.47–6.50) to find an estimate to fields at
            t^{n+1} (described below)
7:       end for
8:       Update estimate to z_D from (6.52)
9:    end for
10: end for
```

Figure 6-2: Solution procedure for a Semi-Lagrangian vertical column model.

outer loop in which the departure points z_D are updated, and an *inner loop* in which the spatially discretised equations are iteratively solved, which is the subject of the next section.

We now linearise these equations about some *reference profiles* (described below), giving us an incremental version of the time-discretised PDEs which can be iterated in the inner loop.

6.3 Linearisation

For each departure point (outer) loop, the equations (6.47–6.50) are solved in an iterative manner, the inner loop, as follows: Consider, after k iterations we have the estimate $\vec{x}^{(k)} = [w, \rho, \theta, \Pi]^{(k)}$ to $[w, \rho, \theta, \Pi]^{n+1}$. We wish to find $\vec{x}^{(k+1)} = [w, \rho, \theta, \Pi]^{(k+1)}$ which solves a residual equation

$$\vec{R}^{(k+1)} := [R_w^{(k+1)}, R_\rho^{(k+1)}, R_\theta^{(k+1)}, R_\Pi^{(k+1)}] = 0\,. \tag{6.53}$$

This can be solved via a Newton method, giving

$$J(\vec{x}^{(k+1)} - \vec{x}^{(k)}) = -\vec{R}^{(k)}\,, \tag{6.54}$$

where J is the Jacobian of $\vec{R}$. The Jacobian can be restrictively expensive to compute in higher dimensions, hence we replace it by a linear matrix $L(\vec{x}^*)$, obtained by linearising $\vec{R}$ about a *reference state* $\vec{x}^*$. We delay discussion of the choice of reference profiles until Subsection 6.3.6. In the ENDGame formulation only the terms important to the stability of the fast gravity waves are retained: e.g. in 2D we would drop the horizontal

terms $(u^{(k+1)} - u^{(k)})\theta_x^*$.

This linearisation results in the linear equation

$$L(\vec{x}^*)(\vec{x}^{(k+1)} - \vec{x}^{(k)}) = -\vec{R}^{(k)}\,, \quad k = 1, 2, \ldots\,. \tag{6.55}$$

We are free to choose $\vec{x}^*$ as we like: In particular, we give an example of a Newton Discretisation with $\vec{x}^* = \vec{x}^{(k)}$, using the most up to date value $\vec{x}^{(k)}$. We also consider a form of the Met Office ENDGame discretisation with $\vec{x}^* = \vec{x}^n$, with the exception of w^*, which is linearised about $w^* = 0$.

The departure points are calculated from the most up-to-date estimates to the winds w_A^{n+1} in the outer loop, then with these departure points, the fields w, θ, ρ and Π are calculated in the inner loop.

We denote the right-hand side of equations (6.47), (6.48) and (6.49) as $(R_w^n)_D$, $(R_\rho^n)_D$ and $(R_\theta^n)_D$ respectively. The left-hand sides form a system of nonlinear PDEs in space (ODEs here, since we are considering only one spatial dimension).

6.3.1 The momentum equation

We now present the linearisation of the time-discretised vertical column equations (6.47-6.50), starting with the semi-Lagrangian momentum equation (6.47)

$$\left[w + \Delta t\alpha\left(c_p\theta\frac{\partial\Pi}{\partial z} + g\right)\right]_A^{n+1} = (R_w^n)_D\,. \tag{6.56}$$

We represent the known right-hand sides of equations (6.47), (6.48) and (6.49) as $(R_w^n)_D$, $\left(R_\rho^n\right)_D$ and $(R_\theta^n)_D$ respectively. As described above we introduce reference states w^*, θ^*, ρ^* and Π^*, such that

$$w = w^* + \widehat{w}\,, \tag{6.57}$$

$$\theta = \theta^* + \widehat{\theta}\,, \tag{6.58}$$

$$\rho = \rho^* + \widehat{\rho}\,, \tag{6.59}$$

and

$$\Pi = \Pi^* + \widehat{\Pi}\,. \tag{6.60}$$

If we linearise the left-hand side of the momentum equation (6.47), thus ignoring all products of terms with hats (the perturbations from the reference states) then we

get the expression

$$\left[w + \Delta t\alpha\left(c_p\theta\frac{\partial\Pi}{\partial z} + g\right)\right]_A^{n+1} \approx w^* + \widehat{w} + \Delta t\alpha c_p\left(\theta^*\frac{\partial\Pi^*}{\partial z} + \theta^*\frac{\partial\widehat{\Pi}}{\partial z} + \widehat{\theta}\frac{\partial\Pi^*}{\partial z}\right), \quad (6.61)$$

where the term $\Delta t\alpha c_p\widehat{\theta}\frac{\partial\widehat{\Pi}}{\partial z}$ has been removed.

For $w^{(k)}$, a given current approximation to w, we want to find a better approximation $w^{(k+1)}$ via an increment w'. With equivalent expressions for ρ, θ and Π, we have

$$w^{(k+1)} = w^{(k)} + w', \quad (6.62)$$
$$\rho^{(k+1)} = \rho^{(k)} + \rho', \quad (6.63)$$
$$\theta^{(k+1)} = \theta^{(k)} + \theta', \quad (6.64)$$
$$\Pi^{(k+1)} = \Pi^{(k)} + \Pi'. \quad (6.65)$$

For our Newton discretisation, we have e.g. $w^* = w^{(k)}$, and as such $w' = \widehat{w}$.

We denote the left-hand side of equation (6.61) for $w^{(k)}$, $\theta^{(k)}$ and $\Pi^{(k)}$ by $L_w^{(k)}$ (or $L_w^{(k+1)}$ with $w^{(k+1)}$ etc.). Then we can find w', the update to w, by looking at the difference between two steps of (6.61) with $w^{(k)}$ and $w^{(k+1)}$, giving

$$w' + \alpha\Delta t c_p\left(\theta^*\frac{\partial\Pi'}{\partial z} + \theta'\frac{\partial\Pi^*}{\partial z}\right) = (R_w^n)_D - L_w^{(k)}. \quad (6.66)$$

We repeat this process for the remaining equations.

6.3.2 The Thermodynamic equation

For the discretised thermodynamic equation (6.49) we have

$$[\theta]_A^{n+1} = [\theta]_D^n. \quad (6.67)$$

Taken as is, this gives an explicit expression for θ^{n+1}, which remains constant within the inner loop. However, later in this chapter we shall permit energy transfer through the inclusion of moisture, hence we construct an incremental form for θ, giving

$$\theta' = (R_\theta^n)_D - L_\theta^{(k)}. \quad (6.68)$$

Currently we have $\theta' = 0$ and θ could be treated as constant in time in the remaining incremental equations, but we include a forcing term in Subsection 6.3.5, giving non-zero θ'.

6.3.3 The continuity equation

For the density equation (6.49),

$$\left[\rho + \Delta t \alpha \rho \frac{\partial w}{\partial z}\right]_A^{n+1} = \left(R_\rho^n\right)_D \,, \tag{6.69}$$

the linearisation and incremental difference, as above, results in the incremental mass equation

$$\rho' + \Delta t \alpha \left(\rho' \frac{\partial w^*}{\partial z} + \rho^* \frac{\partial w'}{\partial z}\right) = \left(R_\rho^n\right)_D - L_\rho^{(k)} \,. \tag{6.70}$$

6.3.4 The State equation

We consider the state equation

$$\left[\Pi^\gamma - \frac{R}{p_0}\theta\rho\right]_A^{n+1} = 0 \,, \tag{6.71}$$

with

$$\gamma := \frac{1-\kappa}{\kappa} \,. \tag{6.72}$$

This equation linearised around reference states becomes

$$\left[(\Pi^*)^\gamma \left(1 + \gamma \frac{\widehat{\Pi}}{\Pi^*}\right) - \frac{R}{p_0}\theta^*\rho^* \left(1 + \frac{\widehat{\theta}}{\theta^*} + \frac{\widehat{\rho}}{\rho^*}\right)\right]_A^{n+1} = 0 \,. \tag{6.73}$$

Dividing by $\Pi^{*\gamma}$ and subtracting the previous inner loop approximation leads to the incremental equation of state

$$\gamma \frac{\Pi'}{\Pi^*} - \frac{R\theta^*\rho^*}{p_0 \Pi^{*\gamma}} \left(\frac{\theta'}{\theta^*} + \frac{\rho'}{\rho^*}\right) = \frac{-L_\Pi^{(k)}}{(\Pi^*)^\gamma} \,. \tag{6.74}$$

6.3.5 Correction Term for the Departure Points

We now discuss departure point correction terms for the momentum, thermodynamic and mass equations. These are small corrections added to the left-hand sides of equations (6.66), (6.68) and (6.70) within the inner loop. They are included in the ENDGame formulation and experience has shown that they are beneficial to the numerical solution procedure (T. Melvin, personal communication). There are a number of ways of justifying their inclusion, [32, §A] and we here present one which feels most "natural" to the author of this thesis.

We consider the momentum equation (6.47), the subject of Subsection 6.3.1. For the

departure point equation (6.52) dependent on w, we can obtain an improved estimate to the departure point by considering the new wind $w^{(k+1)}$,

$$\begin{aligned} z_{DIM} &= z_A - \frac{\Delta t}{2}\left(w^{(l)} + (w^{(k+1)} - w^{(l)}) + w_D^n\right), \\ &= z_D - \frac{\Delta t}{2}(w^{(k+1)} - w^{(l)}), \qquad (6.75) \\ &\approx z_D - \frac{\Delta t}{2} w', \qquad (6.76) \end{aligned}$$

where $w^{(l)}$ is the wind that was used in the outer loop to calculate z_D, and $w' = w^{(k+1)} - w^{(k)}$.

If we consider updated departure point z_{DIM} to be a first order perturbation, then we get an additional term in the right-hand side of the momentum increment equation (6.66),

$$\begin{aligned} (R_w^n)_{DIM} &= \left[w - \beta\Delta t c_p \theta \frac{\partial \Pi}{\partial z}\right]^n_{DIM} \\ &\approx (R_w^n)_D - w' \frac{\Delta t}{2}\left[\frac{\partial w}{\partial z}\right]^n_D . \qquad (6.77) \end{aligned}$$

Noting that, to first order

$$\left[\frac{\partial w}{\partial z}\right]^n_D \approx \frac{\partial w^*}{\partial z}, \qquad (6.78)$$

and assuming α is close to 0.5, we can include this departure point correction term in the left-hand side of the momentum increment equation (6.66), resulting in

$$w' + \alpha\Delta t c_p\left(\theta' \frac{\partial \Pi^*}{\partial z} + \theta^* \frac{\partial \Pi'}{\partial z}\right) + \alpha\Delta t w' \frac{\partial w^*}{\partial z} = (R_w^n)_D - L_w^{(k)}. \qquad (6.79)$$

As we shall see below, in the ENDGame model the wind is linearised about $w^* = 0$, and the correction term in equation (6.79) vanishes.

With a similar argument we arrive at a correction to the mass increment equation

$$\rho' + \alpha\Delta t\left(\rho' \frac{\partial w^*}{\partial z} + \frac{\partial\left(w'\rho^*\right)}{\partial z}\right) = \left(R_\rho^n\right)_D - L_\rho^{(k)}. \qquad (6.80)$$

For the incremental thermodynamic equation, with a θ-correction term we get

$$\theta' + \alpha\Delta t w' \frac{\partial \theta^*}{\partial z} = (R_\theta^n)_D - L_\theta^{(k)}. \qquad (6.81)$$

With this correction term, θ becomes coupled with w, ρ and Π.

We now have our incremental equations to be solved in the inner loop, equations

(6.79–6.81) and the incremental equation of state (6.73). It remains to choose the reference states around which to linearise and to discretise the equations spatially.

We now discuss the choice of reference profiles.

6.3.6 ENDGame and Newton Reference Profiles

As mentioned earlier, in the ENDGame model the model equations are linearised about the pressure, density and temperature profiles from the previous time step, and zero wind, that is

$$w^* = 0\,, \tag{6.82}$$

$$\rho^* = \rho^n\,, \tag{6.83}$$

$$\theta^* = \theta^n\,, \tag{6.84}$$

$$\Pi^* = \Pi^n\,. \tag{6.85}$$

If we also assume that these reference fields satisfy the equation of state then the increment equations with correction terms are

$$w' + \alpha\Delta t c_p \left(\theta' \frac{\partial \Pi^n}{\partial z} + \theta^n \frac{\partial \Pi'}{\partial z}\right) = (R_w^n)_D - L_w^{(k)}\,, \tag{6.86}$$

$$\rho' + \alpha\Delta t \frac{\partial (w'\rho^n)}{\partial z} = \left(R_\rho^n\right)_D - L_\rho^{(k)}\,, \tag{6.87}$$

$$\theta' + \alpha\Delta t w' \frac{\partial \theta^n}{\partial z} = (R_\theta^n)_D - L_\theta^{(k)}\,, \tag{6.88}$$

$$\gamma \frac{\Pi'}{\Pi^n} - \left(\frac{\theta'}{\theta^n} + \frac{\rho'}{\rho^n}\right) = \frac{-L_\Pi^{(k)}}{(\Pi^n)^\gamma}\,. \tag{6.89}$$

This is similar to a 1D form of the vertical slice equations in [32]. The numerical scheme in [56, §5.1] uses a more sophisticated linearisation, which removed the dependence on the reference states at convergence of the inner loop.

The approach we take here is to instead discretise about the most up-to-date esti-

mates to the fields, such that

$$w^* = w^{(k)}\,, \tag{6.90}$$

$$\rho^* = \rho^{(k)}\,, \tag{6.91}$$

$$\theta^* = \theta^{(k)}\,, \tag{6.92}$$

$$\Pi^* = \Pi^{(k)}\,. \tag{6.93}$$

This gives the incremental equations

$$w' + \alpha\Delta t c_p \left(\theta' \frac{\partial \Pi^{(k)}}{\partial z} + \theta^{(k)} \frac{\partial \Pi'}{\partial z}\right) + \alpha\Delta t w' \frac{\partial w^{(k)}}{\partial z} = (R_w^n)_D - L_w^{(k)}\,. \tag{6.94}$$

$$\rho' + \alpha\Delta t \left(\rho' \frac{\partial w^{(k)}}{\partial z} + \frac{\partial \left(w'\rho^{(k)}\right)}{\partial z}\right) = \left(R_\rho^n\right)_D - L_\rho^{(k)}\,. \tag{6.95}$$

$$\theta' + \alpha\Delta t w' \frac{\partial \theta^{(k)}}{\partial z} = (R_\theta^n)_D - L_\theta^{(k)}\,, \tag{6.96}$$

$$\gamma \frac{\Pi'}{\Pi^{(k)}} - \frac{R\theta^{(k)}\rho^{(k)}}{p_0\left(\Pi^{(k)}\right)^\gamma}\left(\frac{\theta'}{\theta^{(k)}} + \frac{\rho'}{\rho^{(k)}}\right) = \frac{-L_\Pi^{(k)}}{\left(\Pi^{(k)}\right)^\gamma}\,. \tag{6.97}$$

As we shall see later in Section 6.5, solving this system requires calculating products and numerical derivatives of the reference states. In the ENDGame formulation, such terms must be calculated once per time step. For our Newton formulation they must be recalculated at every step of the inner loop. Whilst it is not an issue in the case of a 1D vertical column model as presented here, for a full 3D model this would represent a significant computational expense.

A potential compromise could be to update the reference states at every step of the outer loop, using the fields from the previous time for the initial step. We do not discuss this further here.

In the next section, we discuss the spatial discretisation of these equations.

6.4 Spatial Discretisation

We now discretise our equations in space. We again mirror the approach used in ENDGame, with the 3D spatial discretisation from [56, §4]. For Burgers' equation in Chapters 3 and 5 we discretised the solution on a single mesh with points x_i. Here, we use two staggered grids which we express with integer level and half-integer level indices, z_i and $z_{i+1/2}$, known as a Charney-Phillips grid [56, §4.1]. The fields w and θ are represented on integer level grid points, and ρ and Π are represented on half levels. The bottom few levels are shown in Figure 6-3.

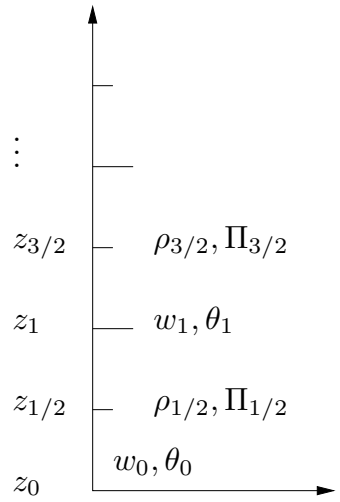

Figure 6-3: First few levels with staggered variables, showing integer levels with w and θ, and half-integer levels with ρ and Π.

We describe two spatial operators, a spatial averaging for stepping between full and half levels, and finite difference operators for approximating derivatives, as described in Sections 3.3 and 5.2. Spatial averaging of a field F at z_i is the weighted average of $z_{i+1/2}$ and $z_{i-1/2}$, defined as

$$\overline{F}^z_i := \frac{z_{i+1/2} - z_i}{z_{i+1/2} - z_{i-1/2}} F_{i-1/2} + \frac{z_i - z_{i-1/2}}{z_{i+1/2} - z_{i-1/2}} F_{i+1/2} \,. \tag{6.98}$$

We use a central difference operator to represent all derivatives,

$$\delta_z(F)_i = \frac{F_{i+1/2} - F_{i-1/2}}{z_{i+1/2} - z_{i-1/2}} \,. \tag{6.99}$$

Similar expressions are used for spatial averaging and representing derivatives at half-integer levels,

$$\overline{F}^z_{i+1/2} := \frac{z_{i+1} - z_{i+1/2}}{z_{i+1} - z_i} F_i + \frac{z_{i+1/2} - z_i}{z_{i+1} - z_i} F_{i+1} \,, \tag{6.100}$$

$$\delta_z(F)_{i+1/2} = \frac{F_{i+1} - F_i}{z_{i+1} - z_i} \,. \tag{6.101}$$

The general approach is to use the spatial averaging operator to ensure the terms are represented on appropriate levels, and then the central difference operator to express any spatial derivatives.

The Newton increment equations are then

$$w_i' + \alpha\Delta t c_p \left(\theta_i' \delta_z \left(\Pi^{(k)}\right)_i + \theta_i^{(k)} \delta_z (\Pi')_i\right) + \alpha\Delta t w_i' \delta_z \left(\overline{w^{(k)}}^z\right)_i = R_{wi}\,, \tag{6.102}$$

$$\rho_{i+1/2}' + \alpha\Delta t \left(\rho_{i+1/2}' \delta_z \left(w^{(k)}\right)_{i+1/2} + \delta_z \left(\overline{\rho^{(k)}}^z w'\right)_{i+1/2}\right) = R_{\rho i+1/2}\,, \tag{6.103}$$

$$\theta_i' + \alpha\Delta t w_i' \delta_z \left(\overline{\theta^{(k)}}^z\right)_i = R_{\theta i}\,, \tag{6.104}$$

$$\left(\gamma \frac{\Pi'}{\Pi^{(k)}}\right)_{i+1/2} - \left(\frac{R\overline{\theta^{(k)}}^z \rho^{(k)}}{p_0 \left(\Pi^{(k)}\right)^\gamma}\right)_{i+1/2} \left(\overline{\left(\frac{\theta'}{\theta^{(k)}}\right)}^z_{i+1/2} + \left(\frac{\rho'}{\rho^{(k)}}\right)_{i+1/2}\right) = R_{\pi i+1/2}\,. \tag{6.105}$$

We now have a set of linear equations to be solved to get a better approximation to the fields w, ρ, θ and Π. The linearised equations are combined to form a Helmholtz equation in terms of the Exner pressure Π, which we solve via finite differences. Finally, the update to Π is back substituted to get expressions for w, ρ and θ. This process, solving for Π constitutes a single step of the inner loop of the solution algorithm given in Figure 2-3.

6.5 Helmholtz Equation

To solve the above linearised equations, we combine them to make a single linear system in terms of $\Pi'_{i+1/2}$ which we can solve. For convenience, we introduce a number of terms dependent only on the reference profiles,

$$H_{0i} = \left(\frac{R\rho^{(k)}\overline{\theta^{(k)}}^z}{p_0(\Pi^{(k)})^\gamma}\right)_i\,, \tag{6.106}$$

$$H_{1i} = \left(1 + \alpha\Delta t \delta_z w^{(k)}\right)_i^{-1}\,, \tag{6.107}$$

$$H_{2i} = \alpha\Delta t c_p \theta_i^{(k)}\,, \tag{6.108}$$

$$H_{\theta i} = n_1 \alpha\Delta t \delta_z \theta_i^{(k)}\,, \tag{6.109}$$

$$H_{wi} = \left(1 + \alpha\Delta t \delta_z \left(\overline{w^{(k)}}^z\right) - \alpha\Delta t c_p H_\theta \delta_z \left(\Pi^{(k)}\right)\right)_i^{-1}\,. \tag{6.110}$$

With these, we can express the discretised incremental equations (6.102–6.105) as

$$w'\left(1+\alpha\Delta t\delta_z\left(\overline{w^{(k)}}^z\right)\right)+\alpha\Delta t c_p\delta_z\left(\Pi^{(k)}\right)\theta'+H_2\delta_z(\Pi')=R_w\,, \tag{6.111}$$

$$H_1^{-1}\rho'+\alpha\Delta t\delta_z\left(\overline{\rho^{(k)}}^z w'\right)=R_\rho\,, \tag{6.112}$$

$$\theta'+H_\theta w'=R_\theta\,, \tag{6.113}$$

$$\gamma\frac{\Pi'}{\Pi^{(k)}}-H_0\left(\overline{\left(\frac{\theta'}{\theta^{(k)}}\right)}^z+\frac{\rho'}{\rho^{(k)}}\right)=R_\pi\,. \tag{6.114}$$

We can eliminate ρ' and θ' by substituting the incremental mass and thermodynamic equations (6.112) and (6.113) into (6.111) and (6.114), leaving us with

$$H_w{}^{-1}w'+H_2\delta_z(\Pi')=R_w-\alpha\Delta t c_p R_\theta\delta_z(\Pi^{(k)}):=\mathfrak{R}_w\,, \tag{6.115}$$

$$\begin{aligned}\gamma\frac{\Pi'}{\Pi^{(k)}}+\alpha\Delta t\frac{H_0H_1}{\rho^{(k)}}\delta_z\left(\overline{\rho^{(k)}}^z w'\right)+H_0\overline{\left(\frac{H_\theta w'}{\theta^{(k)}}\right)}^z\\ =R_\Pi+H_0H_1\frac{R_\rho}{\rho^{(k)}}+H_0\overline{\left(\frac{R_\theta}{\theta^{(k)}}\right)}^z=:\mathfrak{R}_\Pi\,.\end{aligned} \tag{6.116}$$

Next we can substitute equation (6.115) into (6.116) eliminating w', giving

$$\gamma\frac{\Pi'}{\Pi^{(k)}}-D1\left(H_2\delta_z(\Pi')\right)=\mathfrak{R}_\Pi-D1\left(\mathfrak{R}_w\right)=:\mathrm{RHS}\,, \tag{6.117}$$

where the operator $D1$ is defined as

$$D1(X):=\alpha\Delta t\frac{H_0H_1}{\rho^{(k)}}\delta_z\left(\overline{\left(\rho^{(k)}\right)}^z H_wX\right)+H_0\overline{\left(\frac{H_\theta H_w}{\theta^{(k)}}X\right)}^z\,. \tag{6.118}$$

The equation (6.117) forms a linear system of equations which we can solve for Π'. In this 1D model, this system can be solved using a tridiagonal matrix inversion. In higher dimensions solving this system is more computationally expensive. ENDGame uses a stabilised bi-conjugate gradient method [56, §5.3].

6.5.1 Back substitution

We can now recover Π', the inner loop update to the Exner pressure Π. The remaining fields can be updated by back-substituting Π' into the equations used to form the discretised Helmholtz problem (6.117): The increment to the wind, w' can be found from equation (6.115), so that

$$w'=H_w\left(\mathfrak{R}_w-H_2\delta_z\left(\Pi'\right)\right)\,. \tag{6.119}$$

The update to the potential temperature θ' can be found from equation (6.113),

$$\theta' = R_\theta - H_\theta w' \,. \tag{6.120}$$

Finally, the update to the density ρ' can be obtained from equation (6.112),

$$\rho' = H_1 \left(R_\theta - \alpha \Delta t \delta_z \left(\overline{\rho^{(k)}}^z w' \right) \right) \,. \tag{6.121}$$

6.6 Numerical Solutions

We now present the results of the implementation of the SL vertical column model with Newton linearisation, as in the code `main.m` in Appendix A.5. For now, our process will be to define an analytically stationary inversion layer as in Subsection 6.1.1, and run the models, driven exclusively by first the discretisation error, then the discretisation and interpolation errors. We present a number of cases with the different inversion layers for a stationary uniform mesh and for stationary non-uniform meshes from Chapter 4. The non-uniform meshes will be be quadratic, or will equidistribute monitor functions depending on the analytic starting profile.

6.6.1 Static Inversion Layer Calculations

We take samples from an analytically static profile, and hence any wind present in the system is a result of discretisation and interpolation error.

With a current lack of rigorous error analysis, we use the winds produced as an approximate comparison between two meshes, where a lower magnitude of wind is seen as preferable.

The Setup:

As described in Subsection 6.1.1 we prescribe a temperature profile then find θ and Π which analytically satisfy the momentum equation with no wind w. The density ρ is then found from the state equation.

We define our 1D domain from the earth's surface up to 10,000m with a rigid floor and ceiling. Initially we prescribe the temperature as 300K below 4,000m, 320K above 5,000m and varying linearly between 4,000m and 5,000m in the 'inversion layer' (6.28).

These profiles are sampled at 61 vertical points (lowest at 0m, highest at 10,000m) and then used as initial conditions for the SISL method as described earlier in this chapter, summarised in Figure 6-2, with a time step of 60 seconds integrated over 15 time steps.

First we use a uniform discretisation as our benchmark, in Figure 6-4.

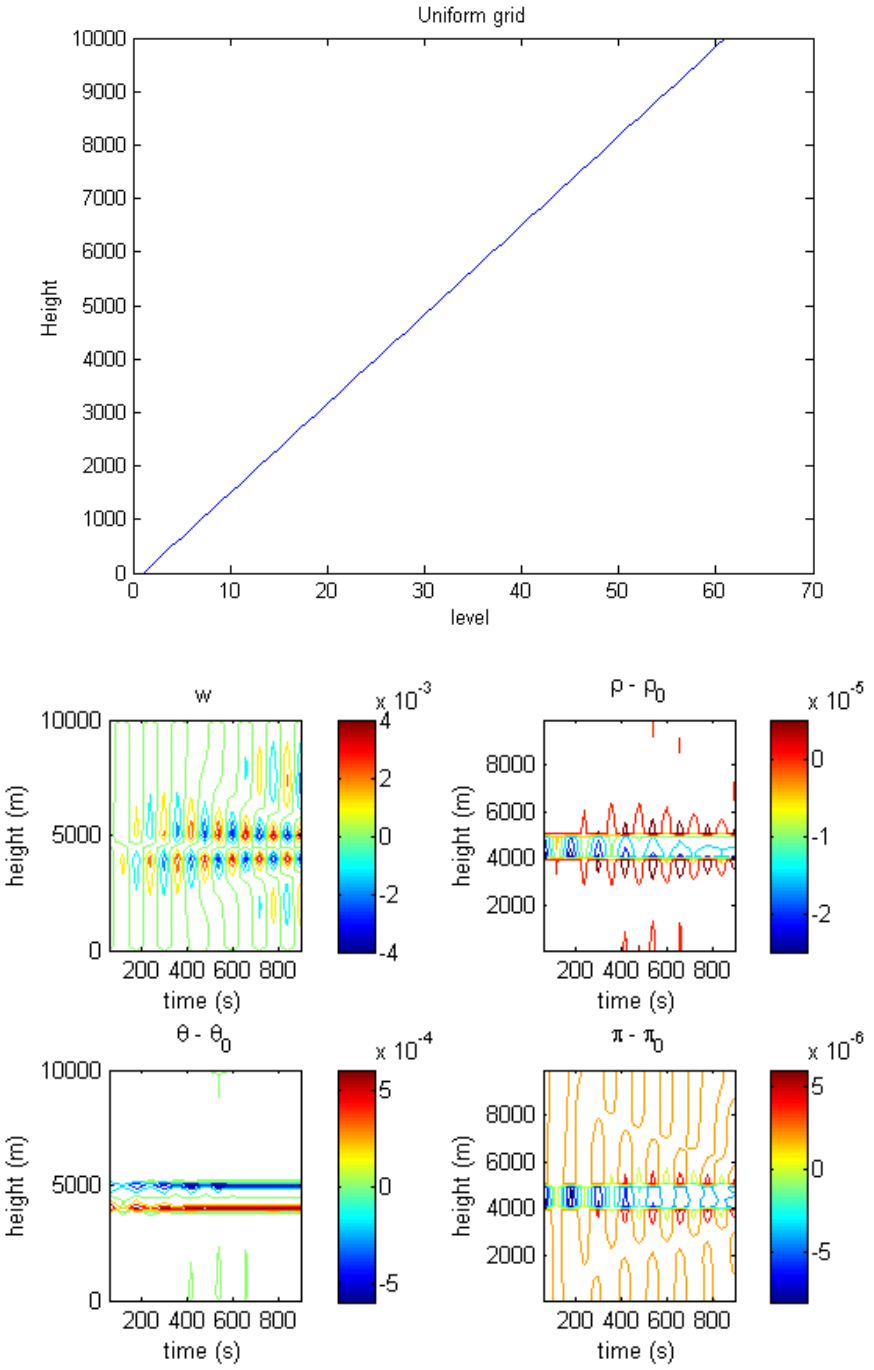

Figure 6-4: Uniform grid

We also look at a quadratic grid with a higher density of levels near the surface, as in Figure 6-5. In the case of the quadratic grid the maximum magnitude of the wind is approximately double of that with the uniform grid.

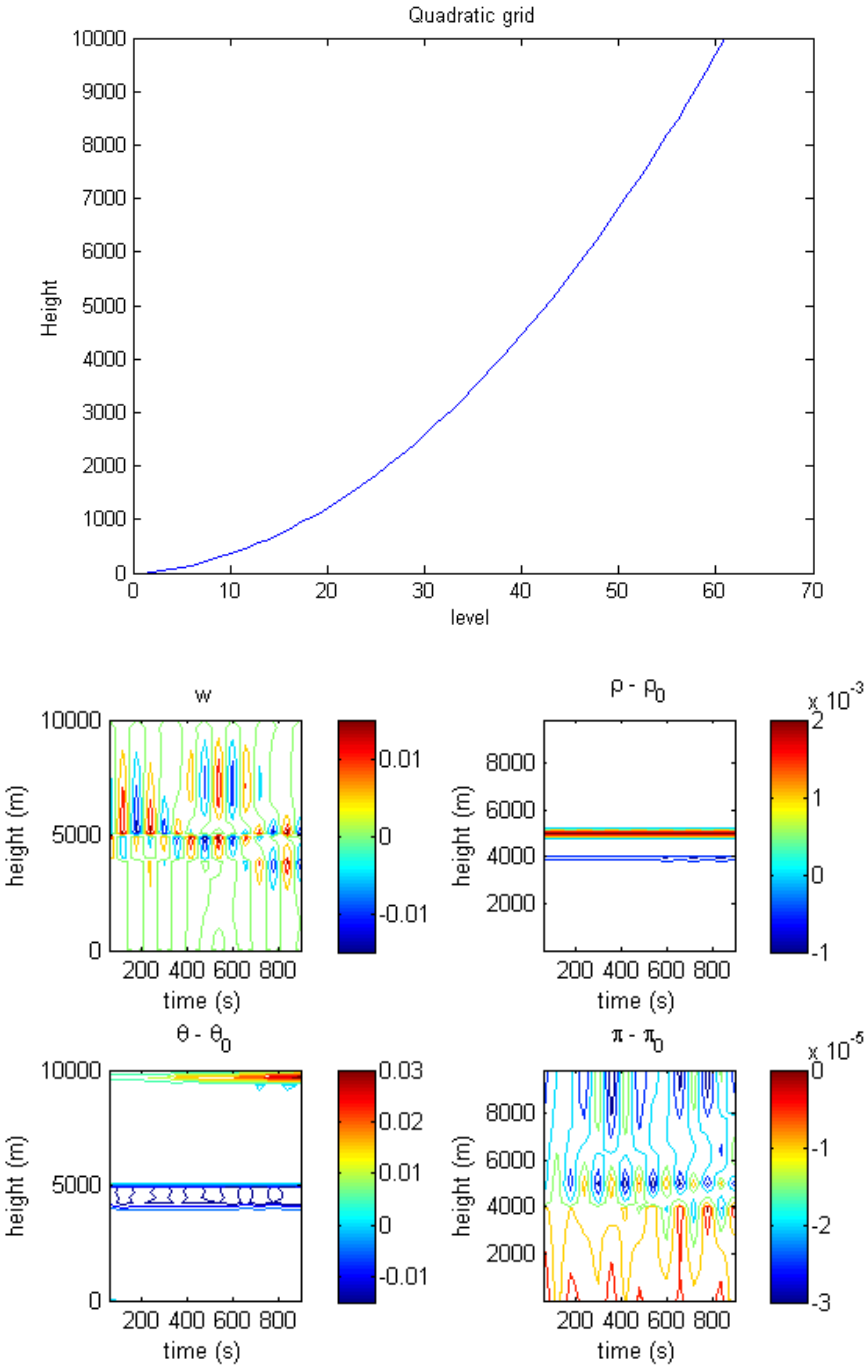

Figure 6-5: Quadratic grid

The next grid, Figure 6-6 is piecewise uniform, in that there are 20 points below 4,000m, 20 in the inversion layer and 20 above. Here the wind is considerably larger, suggesting larger discretisation errors. This is possibly due to the sharp transition in grid-spacing between regions leading to high errors when approximating gradients via finite difference. These problems should be somewhat reduced by applying some runs of smoothing to the monitor function which defines this mesh.

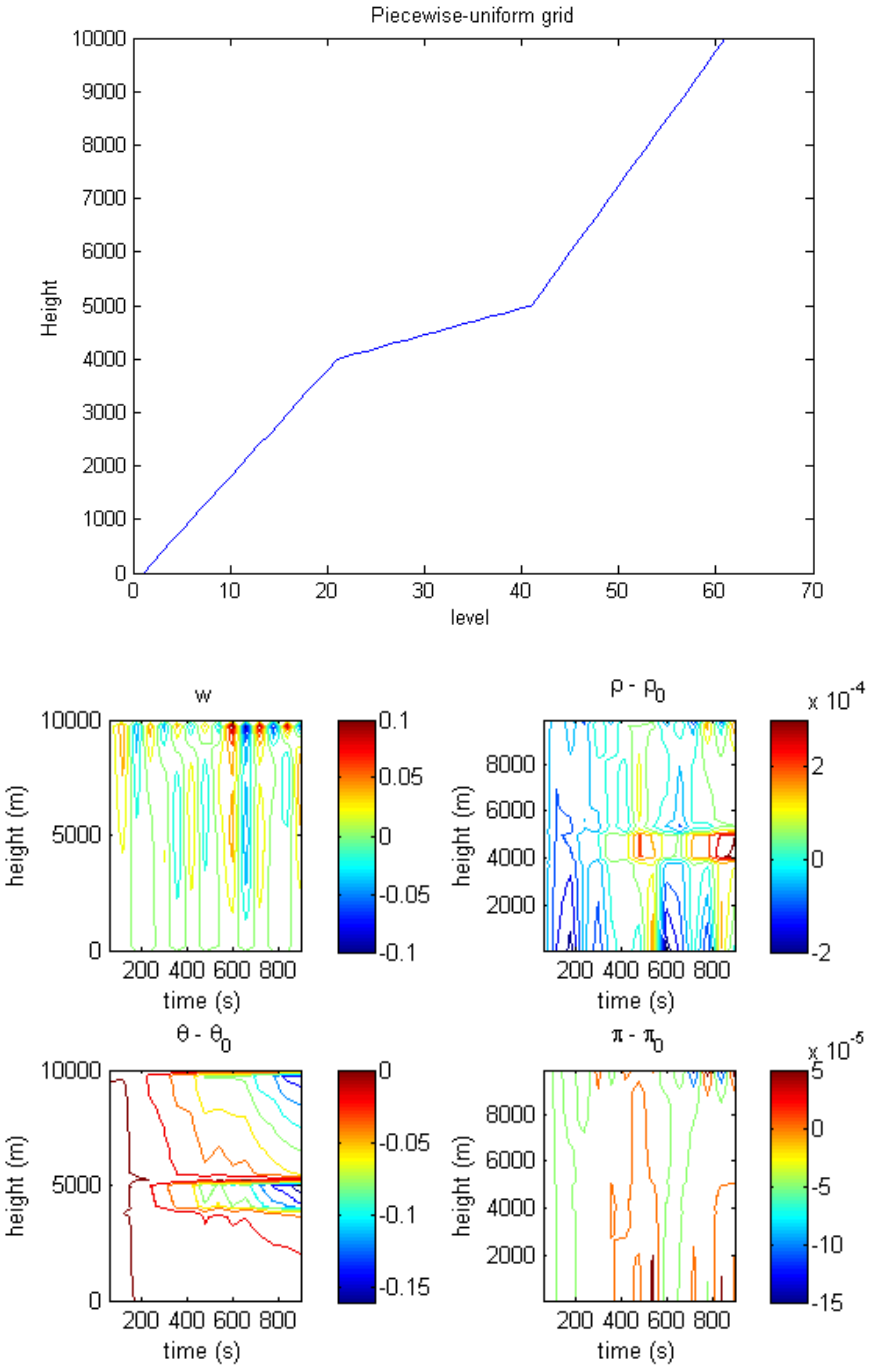

Figure 6-6: Piecewise-linear uniform grid

We look at a mesh adapted using the modified arch-length monitor function

$$M(z) = \sqrt{a + \left(\frac{\partial \theta}{\partial z}\right)^2}\,, \tag{6.122}$$

where a is a small positive parameter scaled with the average value of $\frac{\partial \theta}{\partial z}$, Figure 6-7. The magnitude of the wind is similar to that with a uniform mesh.

Finally Figure 6-8 uses a curvature based monitor functions $M(z) = \sqrt{a + \theta_z z^2}$ but finding the tuning of the parameter a has so far been trial and error. We see that the results produce smaller winds than for the uniform grid. This is due to a smaller starting residual

$$r_w := -gc_p - \theta \frac{\partial \Pi}{\partial z}\,, \tag{6.123}$$

where $\frac{\partial \Pi}{\partial z}$ is calculated via finite difference.

6.7 Moisture

The vertical column model is a useful initial model for testing simple stability properties, but unlike an accurate 3D atmospheric model or Burgers' equation it does not form fronts by itself. We hope to include terms which will add enough interesting dynamics to cause fronts to form in the course of a numerical simulation. To this end, we now include moisture in our vertical column model, following [18, §2.9] and [3].

The amount of water a volume of air can hold can be expressed by the saturation pressure, which increases with temperature. When water vapour condenses into liquid water energy is released as latent heat.

The *specific humidity*, q, is the ratio of the mass of water vapour m_v to total mass, m, in the air,

$$q = \frac{m_v}{m} = \frac{\rho_v}{\rho}\,. \tag{6.124}$$

If the only form of water in the air is water vapour, then the total mass can be separated into water vapour m_v and dry air m_d as

$$m = m_d + m_v\,, \tag{6.125}$$

then

$$q = \frac{m_v}{m_d + m_v}\,. \tag{6.126}$$

Throughout this section subscript d denotes quantities relating to dry air and v to water vapour. Moist air (combined dry and vapour) is denoted either by a subscript

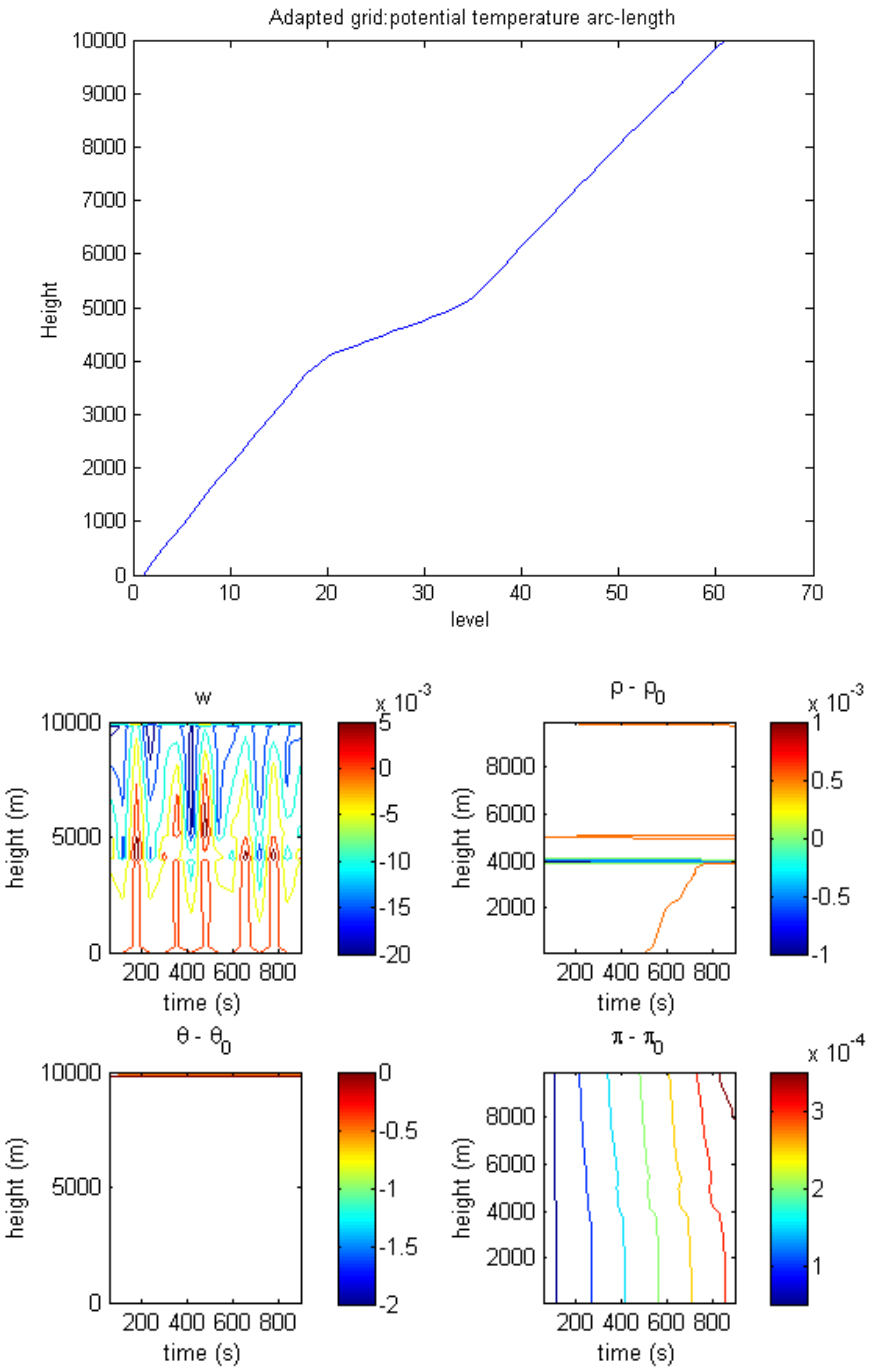

Figure 6-7: Piecewise uniform mesh with smoothing

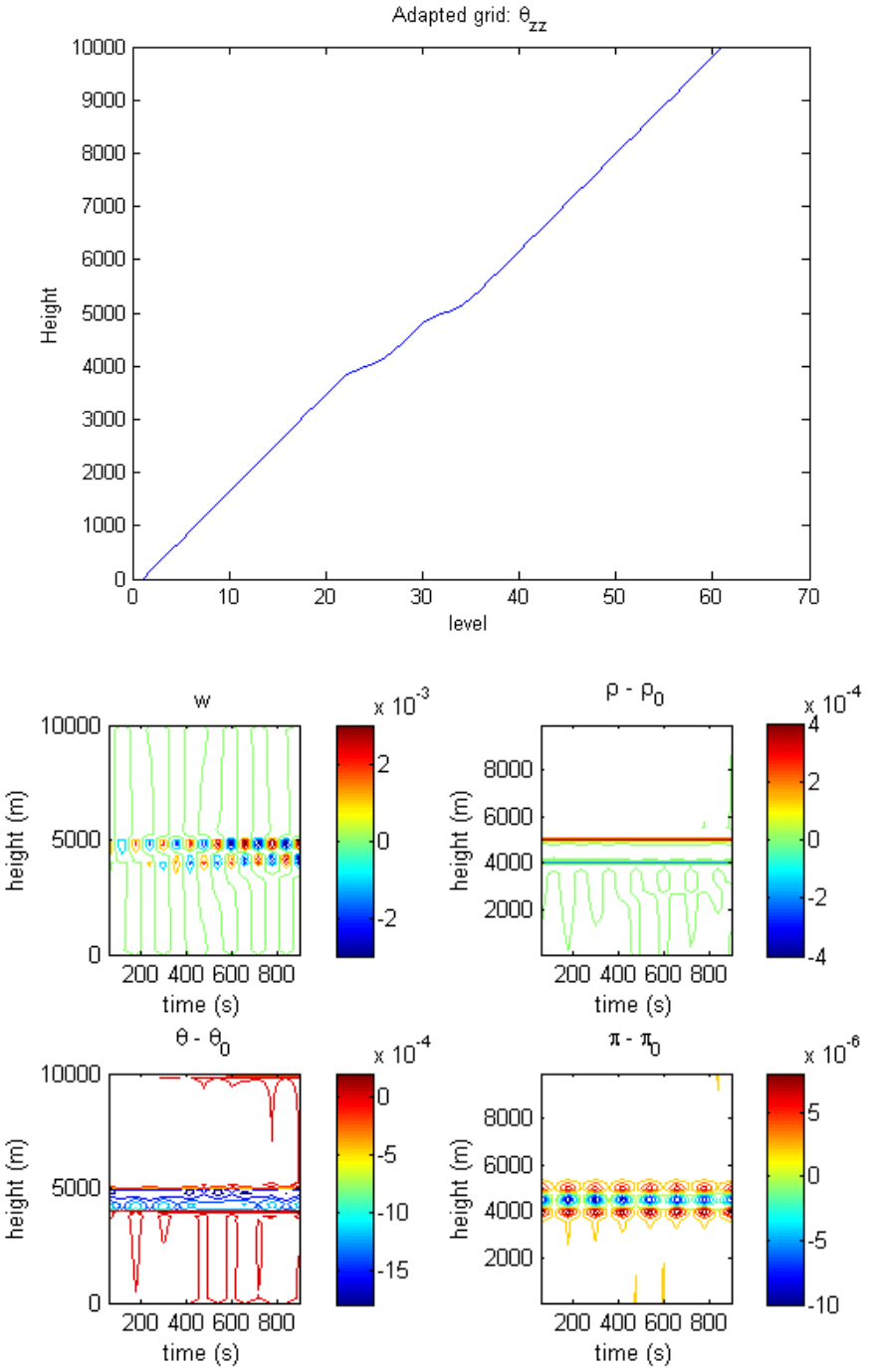

Figure 6-8: Adapted mesh based on potential temperature curvature with smoothing

m, or by no subscript. Quantities relating to cloud or liquid water are denoted by subscripts c and l.

The *vapour mixing ratio* r_v is the mass ratio of vapour to dry air, with

$$r_v = \frac{m_v}{m_d}\,. \tag{6.127}$$

A typical value of r_v in the lower atmosphere is $10\mathrm{g\,kg^{-1}}$, hence q is approximately equal to r_v.

The ideal gas law (6.2) states that

$$p = \rho R T\,, \tag{6.128}$$

where R is the specific gas constant of the given ideal gas, stated to be

$$R = \frac{R^*}{m_{\mathrm{mol}}}\,, \tag{6.129}$$

for R^* the universal gas constant, and m_{mol} the average molecular mass of the gas. *Vapour pressure*, usually denoted by the symbol e, and also satisfies the ideal gas law

$$e = \rho_v R_v T\,. \tag{6.130}$$

With the two equations of state (6.128) and (6.130), we can express the vapour mixing ratio as

$$r_v = \frac{R_d}{R_v}\frac{e}{p-e}\,. \tag{6.131}$$

We denote the molar mass ratio of dry air to water vapour as ε, or

$$\varepsilon := \frac{R_d}{R_v} \approx 0.6226\,, \tag{6.132}$$

and since e is small compared to p, we can approximate equation (6.131) as

$$r_v \approx \varepsilon\frac{e}{p}\,. \tag{6.133}$$

The *saturation pressure* $e_{\mathrm{sat}}(T)$ is the vapour pressure at which air cannot hold any more water vapour. The saturation pressure increases with temperature. If the temperature is reduced such that the vapour pressure would exceed the saturation pressure, water vapour condenses into liquid water droplets, forming clouds. The saturation

pressure has a corresponding saturation mixing ratio r_{vs}, from equation (6.131),

$$r_{vs} = \varepsilon \frac{e_{\text{sat}}}{p - e_{\text{sat}}} \,. \tag{6.134}$$

We can also find an expression for the saturation humidity q_{sat}, which leads to the familiar quantity *relative humidity*, being the ratio between q and q_{sat}.

From [3], the gas constant for moist air is

$$R_m = R_d + r_v R_v \tag{6.135}$$

$$= R_d \left(1 + \frac{r_v}{\varepsilon}\right) , \tag{6.136}$$

and the specific heat at constant volume c_{vml} and constant pressure c_{pml} are given as

$$c_{vml} = c_v + r_v c_{vv} \,, \tag{6.137}$$

$$c_{pml} = c_p + r_v c_{pv} \,. \tag{6.138}$$

The latent heat from vaporisation is

$$L_v(T) = L_{v0} - (c_{pl} - c_{pv})(T - T_0) \,, \tag{6.139}$$

where L_{v0} is a reference latent heat at $T_0 = 273.15\text{K}$,

$$L_{v0} = 2.53106\text{J}\,\text{kg}^{-1} \,. \tag{6.140}$$

Finally the Claussius-Clapeyron relation [18, (2.65)] gives an ODE for the saturation pressure as

$$\frac{\mathrm{d}e_{\text{sat}}}{\mathrm{d}T} = \frac{L_v(T) e_{\text{sat}}}{R_v T^2} \,, \tag{6.141}$$

the solution to which, with reference saturation pressures at 273.15K, is

$$e_{\text{sat}}(T) = e_{\text{sat}}(T_0) \exp\left(-\frac{L_{v0} + T_0(c_{pl} - c_{pv})}{R_v}\left(\frac{1}{T} - \frac{1}{T_0}\right)\right) \cdot \left(\frac{T}{T_0}\right)^{(c_{pv} - c_{pl})/R_v} \,. \tag{6.142}$$

6.8 Moist Vertical Column

We now have expressions for c_{pml} and R_m. We want to modify the vertical column model (6.38–6.41) accounting for these differences when considering moisture. We also

require equations for the transition between water vapour and liquid water (in the form of clouds), and the heat transfer associated with this transition. We express these two forms of water in our model with a water vapour mixing ratio r_v, and a cloud mixing ratio r_c.

We model this as a two-step process, considering a *dynamics step* where the water vapour and cloud are advected unchanged, followed by a *physics step*, where any condensation or evaporation is calculated, with any associated heat transfer.

For the dynamics step we advect the moisture species exactly,

$$\frac{\mathrm{D}r_v}{\mathrm{D}t} = 0\,, \tag{6.143}$$

$$\frac{\mathrm{D}r_c}{\mathrm{D}t} = 0\,. \tag{6.144}$$

For the physics step we want to convert water vapour to cloud, or cloud to water vapour. We specify that cloud is only present when the air is fully saturated, i.e.

$$r_v \in [0, r_{vs}(T)]\,, \tag{6.145}$$

$$r_c > 0 \quad \text{iff} \quad r_v = r_{vs}(T)\,. \tag{6.146}$$

Condensing water vapour releases latent heat, and the saturation mixing ratio r_{vs} is dependent on temperature (and pressure, which is only updated in the dynamics step).

Due to this, we need to find T and r_v such that

$$\begin{aligned} f(T) &= r_{vs}(T) - r_v(T) \\ &= 0\,. \end{aligned} \tag{6.147}$$

We can simplify this problem by assuming that, for the sake of moisture condensation and evaporation, the specific heat of the moist air c_{pml} is close to c_p, and the Latent heat is constant with T,

$$\frac{\mathrm{d}T}{\mathrm{d}r_v} = -\frac{L_{v0}}{c_p}\,. \tag{6.148}$$

The saturation mixing ratio r_{vs} can be calculated for T and p (or θ and Π) from equations (6.134) and (6.142).

We can now solve (6.147) with Newton's method, giving

$$\begin{aligned} T^{(k+1)} &= T^{(k)} - \frac{f(T^{(k)})}{f'(T^{(k)})} \\ &= T^{(k)} + \frac{r_v^{(k)} - r_{vs}(T^{(k)}, p)}{\frac{c_p}{L_{v0}} + \frac{L_{v0} r_{vs}(T^{(k)}, p)}{R_v (T^{(k)})^2}}\,, \end{aligned} \tag{6.149}$$

$$r_v^{(k+1)} = r_v^{(k)} - \frac{c_p}{L_{v0}}(T^{(k+1)} - T^{(k)})\,, \tag{6.150}$$

$$r_c^{(k+1)} = r_c^{(k)} + \frac{c_p}{L_{v0}}(T^{(k+1)} - T^{(k)})\,. \tag{6.151}$$

If $r_c < 0$, then we set

$$r_c = 0\,, \tag{6.152}$$

$$r_v = r_v^{(0)} + r_c^{(0)}\,, \tag{6.153}$$

and

$$T = T^{(0)} - \frac{c_p}{L_{v0}} r_c^{(0)}\,. \tag{6.154}$$

Note that the Newton iteration (6.149) is equivalent to the expression for moisture transfer in [3, (27)], where the authors state that the scheme usually converges in 4 to 6 iterations.

A moisture species has been introduced into the vertical column model. Unfortunately the changes observed with its inclusion seem to be minor. We suggest that the impact of both moisture and vertically adaptive meshes will only play a significant role to the behaviour of the system when the model is extended to a 2D vertical slice model. In 2D there are a number of test problems available [53] on which the vertical adaptivity can be applied to assess more accurately any benefits and downsides to its use.

6.9 Summary, Issues and Reflections

The work presented in this chapter was mostly done whilst at the Met Office. The intention from the outset was to create a basic but Meteorologically relevant model with which to explore the use of moving meshes. There are many options of such a model, and we elected to follow the model formulation of ENDGame, but as a 1D

column.

When producing a forecast, the Met Office have considerable restrictions on computational time. Since we were writing simpler code for research purposes, we had the luxury of using computational techniques which might be seen as too time consuming for operational use. We were able to avoid some of the less obviously justified linearisations and assumption which would lead to faster code, for example by calculating Jacobians less frequently or performing fewer linear solves. **Reverse engineering?**

An issue with the vertical column model is it generally does not lead to clearly developing features. With such features lacking, there is little benefit of a moving mesh over a static mesh adapted to the initial conditions. This lack of movement motivated the inclusion of moisture species into the model. The hope was that this might provide a physically motivated mechanism to increase in energy in the column through the release of latent heat and lead to regions of buoyancy. The formation of such a front not present at initialisation could demonstrate a benefit of having a moving mesh.

Including moisture in the vertical column continuous equations proved to be a more substantial task than initially anticipated due to the subtle interactions between the moisture, temperature and pressure due to the changes to specific heats (which are constants in a dry model).

Furthermore, there were additional complication in implementing the moist model numerically, such as the derivation of the one-step Newton scheme for finding latent heat release and transfer of moisture between vapour and liquid in equations (6.149–6.151). When determining the initial conditions, setting the vapour pressure much above the saturation pressure lead to a large release of latent heat in the first time step. Conversely, having the vapour pressure much below the saturation pressure lead to behaviour which was indistinguishable from the dry model.

In addition to the initial conditions, it raised again the question: was mass - and additionally water - being conserved by the interpolation scheme? The mass conservation was addressed from the ENDGame perspective in a number of publications on SLICE [60, 32], where the conservation of mass equation was expressed in a finite volume form. The mass conservation along with smoothness conditions was then used to define the cubic interpolant of the density ρ. No such form was presented for the moisture species in an ENDGame setting.

This research brought the interpolation scheme into focus. The precise effect of different types of interpolant was not entirely clear in a SISL discretisation, even for the "simple" Burgers' equation. Such realisations lead to the analysis which appears in Chapter 3.

Upon reflection, instead of including moisture in the model, I would have liked to have extended the model to a 2D vertical slice model. Such a model would have in with the $1 + 1$D meshes which were used on artificial inversion layer in Chapter 4, and the associated element quality measures. Such a model would also allow for comparisons with standard test problems from the literature, such as those seen in [32, §4].

Chapter 7

Conclusions

The aim of this thesis was to combine moving mesh methods with semi-Lagrangian (SL) discretisations of PDEs. This was with a view to applying these two techniques simultaneously for applications in numerical weather prediction (NWP).

In Chapter 2 semi-implicit semi-Lagrangian (SISL) methods commonly used in NWP were reviewed. We focused on a two time-level iterated implicit method as developed for the Canadian Climate Model [8], where the departure points are found iteratively from a trapezium rule discretisation as in Cullen (2001) [9], and spatial derivatives are discretised as finite differences, as currently in use by the Met Office. A number of interpolants were described, along with methods for ensuring local monotonicity such that an interpolant does not create new maxima or minima. We then introduced Burgers' equation, which was chosen as a test problem, which can be expressed with a Lagrangian derivative. In particular, we discussed the travelling wave solutions of Burgers' equation, which move with a wave speed c, a front of height 2α and a front width of order $\mathcal{O}(\varepsilon/\alpha)$, where c and α are determined by the boundary conditions, and ε is the viscosity parameter in Burgers' equation.

In Chapter 3 we explored the numerical solutions of Burgers' equation with a SL discretisation. We found the exact solutions of the Lagrangian trajectories and presented the SL discretisation of Burgers' equation in the style of the SL methods used at the Met Office. We showed preliminary numerical experiments with linear interpolation, observing that the numerical solution resembles an exact travelling wave solution with correct front height, but different numerical wave speed $\widehat{c}$ and width, with the width expressed in terms of a numerical viscosity parameter $\widehat{\varepsilon}$. Next we analysed the

SL discretisation of Burgers' equation by forming a modified equation and considering expansions of the numerical solution and wave speed in terms of α. We arrived at terms for $\widehat{c}$ and $\widehat{\varepsilon}$, valid under the assumption that, for given spatial mesh and time step, the integer part of the Courant number does not change in space. These were shown to be in good agreement with numerical results for linear interpolation, as analysed, and for other interpolants, and showed qualitatively similar results for larger α, but only for small Courant number. This analysis could be extended for higher order interpolants, such as cubic Lagrange interpolants.

In Chapter 4 we presented moving meshes as time-dependent maps with associated mesh densities, or monitor functions. We discussed the equidistribution principle to find a 1D mesh for a given monitor function (including a simple algorithm where one assumes a piecewise linear monitor function) and moving mesh PDEs (MMPDEs). We discussed 2D spatial meshes, uniform in one dimension and adapted in the other, which we refer to as 1+1D meshes, showing some measures which could be employed to characterise elements of such meshes. Finally we discussed the process of smoothing monitor functions, and averaging a monitor function with a uniform monitor function, processes commonly employed before applying the equidistribution principle.

In Chapter 5 we implemented moving meshes with Burgers' equation. We first discussed finite differences on non-uniform meshes and energy preserving discretisations of the advection term in inviscid Burgers' equation, which we use for the corresponding term in Burgers' equation. We described two methods for coupling Eulerian discretisations of PDEs with moving meshes: static rezoning, where we remesh between time steps, interpolating the solution onto the new mesh, and a change of coordinates for a moving mesh, where the PDE is modified to account for these moving mesh points and is solved in a computational domain. For the static rezoning the mesh points were moved via the equidistribution principle and the PDE, Burgers' equation, was solved with a θ-method. For the moving frame of reference, the mesh movement was dictated by a MMPDE which was coupled with the discretised Burgers' equation and the two solved simultaneously using MATLAB's implementation of ODE45. Numerical results were shown with sine-based initial conditions. For the moving mesh semi-Lagrangian (MMSL) method, the mesh points were moved via the equidistribution principle. Compared to the SL method, the MMSL method required no additional interpolation. Numerical results were presented for the sine-based initial conditions, giving good comparisons with the other moving mesh techniques in Chapter 5, and for

the travelling wave initial conditions, showing small errors for the numerical viscosity $\widehat{\varepsilon}$ and the front speed $\widehat{c}$ compared to the uniform meshes in Chapter 3. Future work could be done looking at the MMSL method applied to PDEs with two spatial dimensions, with an obvious first step being 2D Burgers' equation. Investigating the 1+1D meshes described in Chapter 3 would be of particular interest in a NWP setting, since it could lead on to a vertical slice model.

In Chapter 6 we presented model equations for a dry vertical column in one spatial dimension, and corresponding hydrostatic profiles, including inversion layers. This problem was chosen to be more realistic of a meteorological calculation than Burgers' equation, and threw up many new challenges. The model was discretised in a semi-Lagrangian fashion. This was more complicated than Burgers' equation, since the system to be solved for given departure points is nonlinear (in contrast to Burgers' equation, which results in a linear system). To solve it a linearisation was used based on a combination of that used by the Met Office and a Newton-Raphson algorithm. We carried out numerical experiments where the initial conditions were sampled from a hydrostatic inversion layer profile. In Burgers' equation the errors arising from a poor resolution were mostly diffusive. In the vertical column the errors result in the production of spurious waves. If the mesh used is adapted to a curvature based monitor function then these waves are reduced. In order to create more movement in the model we modified the equations to model a moist vertical column, where the change of water vapour into liquid water releases heat. The algorithm for calculating the maximum mass of water vapour a volume of air can hold were rederived from the literature. The numerical experiments on these two models raised questions that motivated the inclusion of the interpolation in the more careful analysis in Section 3.4. More work is required in order to better understand the errors present in numerical experiments, such as the effect of the mesh on conservation of various terms, before further work can be down on coupling these discretised models with moving meshes. These models could then be extended to vertical slice models with two spatial dimensions (one horizontal and one vertical), for which there are a number of published test problems, which could be used to investigate 1D adaptivity in a meteorologically relevant model.

Appendix A

Code

A.1 burg2.m

```
function [Un,X_A,bigXstar] = burg2(N, Nt, param_file)
% A light-weight SL 1D burgers' solver, uniform grid, linear interpolation
%
% [U,X] = burg2(N,Nt) returns the final solution to a semi-Lagrangian
% method for viscous Burgers' equation with N internal spatial points and
% Nt time steps for the experiment described in Section 3.3
%
% [U,X,Xstar] = burg2(N,Nt) is as above, but returns the location of
% x_star for all time steps, the point such that U(x_star, t) = c. Calls a
% simple external function get_m_x (not included).
%
% [U,X] = burg2(N, Nt, param_file) is as above, but loads problem
% parameters from a .mat file with filename specified by param_file.
%
% This is a simplified version of the code used for moving mesh
% semi-Lagrangian Burgers' equation in Chapter 5. Main changes required
% involve recalculating A, B and C when the mesh moves (require an old B
% for one time step), interpolation of U onto the new mesh and saving the
% mesh points to be output.

% Stephen P. Cook 03-04-2016

old_path = addpath([pwd,'\options']);

% Load default model parameters, see Section 3.3
load('params_default',...
  'K', ...                                % Number of outer loops
```

```
  'theta_u', 'theta_x', 'epsilon', ...   % Problem parameters
  'x_l', 'x_r', 't0', 'tmax',...         % Domain parameters
  'c', 'alpha_0', 'u0', 'u_l', 'u_r',... % Initial and Boundary Conditions
  'plotting', 'plotlims');               % Plot options
if nargin==3
  load(param_file);
end

if nargout==3 % Whether to create bigXstar or not
  track_front = 1;
else
  track_front = 0;
end

% Setup
Dx = (x_r-x_l)/(N+1);
Dt = (tmax-t0)/Nt;

X_A_bc = (x_l:Dx:x_r)';
X_A = X_A_bc(2:end-1);

% Finite difference matrix and problem specification.
delta2 = 1/(Dx^2) * ...
    (-2*eye(N) + diag(ones(N-1,1),1) + diag(ones(N-1,1),-1));
% Can express SL Burgers' equation as
%   A*u(t(n+1)) + theta_u*C = [B*u(t(n)) + (1-theta_u)*C]_D
% (see Figure 3-1)
A = eye(N) - Dt*epsilon*theta_u*delta2;
Ainv = A^(-1);
B = eye(N) + Dt*epsilon*(1-theta_u)*delta2;
C = Dt*epsilon/(Dx^2)*[u_l;zeros(N-2,1);u_r];

% Initialisation
Un = u0(X_A);
U_out = zeros(N,Nt);     % Can be used to track solution
X_D_out = zeros(N,Nt,K); % Can be used to check departure point convergence
Unplus1 = Un;
X_D = X_A - Dt*Unplus1;
if track_front
  bigXstar = zeros(Nt+1,1);
end % if track_front

for tt = 1:Nt % Time Loop
  for kk = 1:K % Outer Loop
    % Calculate departure points
    X_D_old = X_D;
```

```
    X_D = X_A - Dt*Unplus1;
    X_D(X_D<x_l) = x_l;
    X_D(X_D>x_r) = x_r;
    for ll = 1:2 % Departure point iteration, see Section 2.4
      Un_D = interp1(X_A_bc,[u_l;Un;u_r],X_D);
      X_D = X_A - Dt*(theta_x*Unplus1 + (1 - theta_x)*Un_D);
      X_D(X_D<x_l) = x_l;
      X_D(X_D>x_r) = x_r;
    end % for ll
    R_D = interp1(X_A_bc,[u_l;B*Un + (1-theta_u)*C;u_r],X_D);
    %ENO_pp = interp_ENO(X_A_bc,[u_l;B*Un + (1-theta_u)*C;u_r]);
    %R_D = ppval(ENO_pp, X_D);
    % Solve the SL system (tridiagonal, could use Thomas algorithm)
    Unplus1 = Ainv*(R_D + theta_u*C);
    X_D_out(:,tt,kk) = X_D;
    X_D_diff = X_D - X_D_old;
  end % for kk
  Un = Unplus1;
  U_out(:,tt) = Unplus1;
  if track_front
    [~,bigXstar(tt+1)] = get_m_x(Un,X_A,c,alpha_0);
  end % if track_front
end % for tt

if plotting
  for tt = 1:Nt
  plot(X_A_bc,[u_l;U_out(:,tt);u_r])
  ylim(plotlims)
  drawnow
  end
end % if plotting

path(old_path);
end % function burg2
```

A.2 equidistribute.m

```
function Y = equidistribute(x0,X,M)
% Find points xout that equidistribute the lin. interp. of (X,M)
%
% Y = equidistribute(x0,X,M) finds the points which equidistribute the
% linear interpolant of (X,M) with the same shape as x0. Assumes that X is
% ordered with unique values, (as such we should have Y(1) = X(1) and
```

```
% Y(end) = X(end)), that length(X) == length(M) and that all(M>0).

% Stephen Cook 03-04-2016

Y = zeros(size(x0));
II = length(Y) - 1;
JJ = length(X) - 1;

%
```

$$intMi(i) = \int_{X(i)}^{X(i+1)} m(x)dx,$$

```
%
```

$$intM(i) = \int_{X(1)}^{X(i)} m(x)dx$$

```
% and
%
```

$$theta = \int_{X(1)}^{X(II)} m(x)dx$$

```
% Assuming M is piecewise linear.

% Integrate (X,M) with the trapezium rule
intMi = 1/2*(M(1:end-1)+M(2:end)).*(diff(X));
intM = [0;cumsum(intMi(:))];
theta = intM(end);

Y(1) = X(1);
% jj tracks which interval of X we are in
jj = 1;
for ii = 2:II
  Target = (ii - 1)/II * theta;
  % Find the interval [X(jj), X(jj+1)] in which Y(ii) lies.
  while and(intM(jj) < Target, jj<(JJ))
    jj = jj + 1;
  end % while
  jj = jj - 1;

  XL = X(jj);
  ML = M(jj);
  XR = X(jj+1);
  MR = M(jj+1);
  target_local = Target - intM(jj);
  mx = (MR-ML)/(XR-XL);

  % We want to find Y such that
```

```
  %
```

$$\int_{XL}^{Y} M(x)dx = target\_local.$$

```
  % In this interval
  %
```

$$M(x) = mx * (x - XL) + ML.$$

```
  % so we integrate and use the quadratic formula to get
  %
```

$$Y = XL + (-ML + sqrt(ML^2 + 2 * mx * target\_local))/mx.$$

```
  % For small m, this gives huge error, so we rearrange as
  Y(ii) = XL + 2*target_local/(ML + sqrt(ML^2 + 2*mx*target_local));
end % for ii

Y(II+1) = X(JJ+1);

end % function equidistribute
```

A.3 burgers.m

```
function [bigU,bigX] = burgers(N, Nt, param_file)
% Eulerian theta method for Burgers' equation with moving mesh
%
% [bigU,bigX] = BURGERS(N,Nt) performs a theta method integration using
% the default boundary and initial conditions from param_default.mat,
% returning Nt-by-N matrices containing the solution and mesh over time.
% See Section 5.2.
%
% [bigU,bigX] = burgers(N, Nt, param_file) is as above, but loads problem
% parameters from a .mat file with filename from param_file.
%
% Uses included subfunctions mk_fd_matrices and mk_plot. Also uses a
% function not included, move_mesh.m which forms a smoothed monitor
% function M defined at X_, then calls equidistribute(X_,X_,M)

% Stephen P. Cook 11-10-2016

old_path = addpath([pwd,'\options']);

% Load default model parameters
load('params_default',...
  'NewtonIts', 'NewtonTol',...          % Newton iterations and tolerance
  'theta_u', 'epsilon', ...             % Problem parameters
```

```
  'x_l', 'x_r', 't0', 'tmax',...          % Domain parameters
  'c', 'alpha_0', 'u0', 'u_l', 'u_r',... % Initial and Boundary Conditions
  'plotting', 'plotpause', 'plotlims');  % Plot options
if nargin==3
  load(param_file);
end


Dx = (x_r - x_l)/(N+1);
Dt = (tmax-t0)/Nt;

mopts = mk_move_opts('Exact2');

X_ = (x_l:Dx:x_r)';

X = X_(2:end-1);

[del2i, del_c, ave, b_del2, b_del_c, b_ave] = mk_fd_matrices(X_, u_l, u_r);

bigU = zeros(Nt, N);
bigX = zeros(Nt, N);
T = Dt*(1:Nt)';

% The equation to be solved is
%   u_t = -(ave*U).*(del_c*U) + epsilon*(del2i*U)
% via the theta method. For theta > 0, need to solve
%   F(U) = U - Dt*theta*(epsilon*del2i*U - (ave*U).*(del_c*U)) - RHS = 0
%   RHS = U_old + ...
%         Dt*(1-theta)*(epsilon*del2i*U_old - (ave*U_old).*(del_c*U_old))
% JACOBIAN OF F(U)
%   J(U) = I - Dt*theta*epsilon*del2i + J2(U)
%   J2(U) = Dt*theta*diag(ave*U)*del_c + diag(del_c*U)*ave

% Initiation
U_old = u0(X);

% U independent part of Jacobian
% (Independent of time if using a static mesh)
J1 = eye(N) - Dt*theta_u*epsilon*del2i;
for ii = 1:Nt % timeloop
  % Forward euler, initial estimate
  U = U_old + Dt*(epsilon*(del2i*U_old + b_del2)...
      - (ave*U_old + b_ave).*(del_c*U_old + b_del_c));
  if theta_u % if theta == 0 then stop at the initial estimate, else Newton
  RHS = U_old + ...
      Dt*(1-theta_u)*(epsilon*(del2i*U_old + b_del2)...
```

```
      - (ave*U_old + b_ave).*(del_c*U_old + b_del_c));
    for jj = 1:NewtonIts % Newton loop
      F = U - Dt*theta_u*(epsilon*(del2i*U + b_del2)...
          - (ave*U + b_ave).*(del_c*U + b_del_c)) - RHS;
      % U dependent part of Jacobian (so J_F(U) = J1 + J2(U))
      J2 = Dt*theta_u*(diag(ave*U + b_ave)*del_c + diag(del_c*U + b_del_c)*ave);
      % Newton solve. Tridiagonal solve to come later.
      U_prime = - (J1 + J2)\F;
      U = U + U_prime;
      if (max(abs(U_prime)) < NewtonTol*U)
        break
      elseif (jj==NewtonIts)
        warning(sprintf('Newton method not converged after %d iterations',jj))
      end
    end % for jj, Newton loop
  end % if theta_u
  % Store output
  bigU(ii,:) = U;
  bigX(ii,:) = X;
  % Remesh every time step
  X_old = X_;
  % New mesh
  X_ = movemesh(mopts,X_,[u_l;U;u_r]);
  X = X_(2:end-1);
  % Interpolate U onto this new mesh
  % Currently Linear interpolation, not limited.
  U = interp1(X_old,[u_l;U;u_r],X);
  % Recalculate the FD matrices
  [del2i, del_c, ave, b_del2, b_del_c, b_ave] = mk_fd_matrices(X_, u_l, u_r);
  % Recalculate the U independent part of Jacobian
  J1 = eye(N) - Dt*theta_u*epsilon*del2i;
  U_old = U;
end % ii, timeloop

if plotting
  % Plot the solution
  figure(1)
  mk_plot(bigX, bigU, T, x_l, x_r, u_l, u_r, plotpause, plotlims);
  % Plot the mesh movement
  figure(2)
  for ii = 1:length(X)
      plot(bigX(:,ii),T)
      hold on
  end
  hold off
end % if plotting
```

```
path(old_path);
end % function main

function [del2i, del_c, ave, b_del2, b_del_c, b_ave] =...
    mk_fd_matrices(X_, u_l, u_r)
%MK_FD_MATRICES Makes finite difference matrices and BC vector terms
%
% [D_2,D_c,Ave,b_2,b_c,b_ave] = MK_FD_MATRICES(X_, u_l, u_r) creates the
% second order finite difference matrix D_2, the centred finite difference
% matrix D_c and a tridiagonal averaging matrix Ave for a mesh X_, and
% Dirichlet boundary condition correction vectors for endpoints
% u(X_(1))=u_l and u(X_(end)) = u_r.

Dx = diff(X_);
N = length(Dx) - 1;

del2i = zeros(N);
del_c = zeros(N);
%ave = zeros(N);

b_del2 = zeros(N,1);
b_del_c = zeros(N,1);
b_ave = zeros(N,1);

for ii = 2:N-1
  denom = 0.5*Dx(ii)*Dx(ii+1)*(Dx(ii)+Dx(ii+1));

  % Irregular second order finite difference operator.
  del2i(ii,ii-1) = Dx(ii+1)/denom;
  del2i(ii,ii) = -(Dx(ii) + Dx(ii+1))/denom;
  del2i(ii,ii+1) = Dx(ii)/denom;

  % Irregular centred difference operator.
  del_c(ii,ii-1) = -1/(Dx(ii) + Dx(ii+1));
  del_c(ii,ii+1) =  1/(Dx(ii) + Dx(ii+1));
end % for ii
% Left boundary, U(X[0]) = u_l
del2i(1,1) = -2/(Dx(1)*Dx(2));
del2i(1,2) = 1/(0.5*Dx(2)*(Dx(1)+Dx(2)));
del_c(1,2) = 1/(Dx(1) + Dx(2));

% Right boundary, U(X[N+1]) = u_r
del2i(N,N) = -2/(Dx(N)*Dx(N+1));
del2i(N,N-1) = 1/(0.5*Dx(N)*(Dx(N)+Dx(N+1)));
del_c(N,N-1) = -1/(Dx(N) + Dx(N+1));
```

```
% Unweighted averaging operator over 3 values.
ave = 1/3*(eye(N) + diag(ones(N-1,1),1) + diag(ones(N-1,1),-1));

% Boundary correction terms
b_ave(1) = u_l/3;
b_ave(N) = u_r/3;

b_del_c(1) = -u_l/(Dx(1) + Dx(2));
b_del_c(N) =  u_r/(Dx(N) + Dx(N+1));

b_del2(1) = u_l/(0.5*Dx(1)*(Dx(1)+Dx(2)));
b_del2(N) = u_r/(0.5*Dx(N+1)*(Dx(N)+Dx(N+1)));
end % function mk_fd_matrices

function mk_plot(bigX, bigU, T, x_l, x_r, u_l, u_r, plotpause, plotlims)
%MK_PLOT Creates a moving plot from matrix stored results
%
% MK_PLOT(bigX, bigU, T, x_l, x_r, u_l, u_r, plotpause, plotlims) plots the
% data stored in bigX and bigU with endpoints [x_l x_r] and [u_l u_r]
% respectively for all t in T.
Nt = length(T);
  for ii = 1:Nt
    plot([x_l,bigX(ii,:),x_r],[u_l,bigU(ii,:),u_r])
    ylim(plotlims)
    title(['t = ',num2str(T(ii))])
    drawnow()
    pause(plotpause)
  end % for ii
end % function mk_plot
```

A.4 fd_adaptive.m

```
function [tout,uout]=fd_adaptive(N)
% Function for solving Burgers' Equation with a moving mesh
% Stephen Cook, 02-04-2012

% For solving the PDE
%     u_t - (u_xi/x_xi)*(x_t - u) -
%                         epsilon/x_xi*(u_xi/x_xi)_xi = 0,
%     x_t - 1/tau*(x_xi*M(x))_xi = 0,
% with a monitor function M(x), here being
%     M = sqrt(1+b.*(u_xi./x_xi).^2).
```

```
%
% -INPUT-
%  N - The number of points required in the computational
%       domain.
%
% -OUTPUT-
%  tout - The times corresponding to columns of uout
%  uout - The calculated solution, 2*N by t_N matrix.
%      Single column corresponds to [u;x] at time
%      given in tout.
%
% -PARAMETERS-
% epsilon  - In PDE, gives order of the width on which we
%            have rapid change for t > 0.
% tau      - Relaxation factor for the moving mesh PDE.
% b        - A parameter for the monitor function.
% implicit - Whether to use ODE15i or ODE15s.
% t_N      - Number of timepoints to be outputted.
% t_max    - Maximum time to run solution up to.
%
% -VARIABLES-
% xi - Var in computational domain
% x  - Var in physical domain
% u  - Discretised Solution
% t  - Time
% y  - Stacked vector, y=[u;x]
%
% -SUBFUNCTIONS-
%    x   = igrid(xi)         - Initial grid.
% [C,F,B]= mk_fd_matrices()  - Finite Difference Matrices.
%    M   = monitor(u,up,x,xp)- Monitor Function.
%  utout = ut(y[,yp])        - Burgers' equation.
%  xtout = xt(y)             - Moving Mesh PDE.
%   Mout = outputM(uout)     - Monitor function for uout.
%          myplot(uout, tout)- Plotting function.

if nargin==0
    N=17;
end % if nargin
if nargout==0
end

epsilon= 2e-3; % Parameters
tau= 1;
b= 1;
implicit= false;
```

```
t_max=1.5;
t_N= 151;  % Points to plot

h= 1./(N-1);       % Computational grid width
xi= h*((1:N)-1)'; % Computational grid

x0 = igrid(xi);    % Initial Physical Grid

y0(N)= 0;
y0 = [sin(2*pi*x0)+ 1/2*sin(pi*x0);x0]; % Initial Conditions.
yp0 = zeros(2*N,1);

basepp= basemonitor(igrid(xi)); % Call this so it is a GLOBAL var.
% Main ODE solver, MATLAB inbuilts
if implicit
    f= @(t,y,yp) yp - [ut(y,yp);xt(y)];
    [tout,uout]= ode15i(f,t_max*(0:(t_N-1))./(t_N-1),y0,yp0);
else
    f= @(t,y) [ut(y);xt(y)];
    [tout,uout]= ode15s(f,t_max*(0:(t_N-1))./(t_N-1),y0,yp0);
end % if implicit

myplot(uout, tout); % Plot output.

%  ----------------END OF MAIN FUNCTION----------------
%
% SUBFUNCTIONS

function base= igrid(xi)
% Subfunction for defining the initial grid at t=0.
base= xi;
end % function igrid

function [C,F,B]= mk_fd_matrices()
% Subfunction for making the finite difference matrices.

C= diag(ones(N-1,1),1);
C= C - C';
C(1,1)= -2;
C(1,2)= 2;
C(N,N-1)= -2;
C(N,N)= 2;
C = C./(2*h);

F= (diag(ones(N-1,1),1) - diag(ones(N,1)))./h;
B= -F';
```

```
F(N,:)= F(N-1,:);
B(1,:)= B(2,:);
end % function mk_fd_matrices

function M= monitor(u, x)
% Monitor function + smoothing

% Use the switch parameter mtype to choose the monitor
% function,
%  1 - Arclength,
%      M = sqrt(1 + b*u_x^2)
%  2 - Mod'd Arclength (c=b for now).
%      M = sqrt(1 + b*u_x^2 + c*u_xx^2)
mtype= 1;
mnormalise= 1; % Normalise or not

up= (N+1)*[2*(u(2)-u(1));...
    u(3:end)-u(1:end-2);...
    2*(u(end)-u(end-1))];
xp= (N+1)*[2*(x(2)-x(1));...
    x(3:end)-x(1:end-2);...
    2*(x(end)-x(end-1))];
if mtype==1
    M= sqrt(1+b.*(up./xp).^2);
elseif mtype==2
    upp= (N+1)^2*[u(3)-2*u(2)+u(1);...
        u(3:end)-2*u(2:(end-1))+u(1:(end-2));...
        u(end-2)-2*u(end-1)+u(end)];
    xpp= (N+1)^2*[x(3)-2*x(2)+x(1);...
        x(3:end)-2*x(2:(end-1))+x(1:(end-2));...
        x(end-2)-2*x(end-1)+x(end)];
    M= sqrt(1+b.*((up./xp).^2 + ...
    (upp./(xp.^2)-up.*xpp./(xp.^3)).^2));
end % if

% Normalise
if mnormalise
    %trapz(x,M)
    M=(M/trapz(x,M));
end

% Smoothing (MATLAB inbuild, 5 point smoothing)
M=smooth(M);
end % function monitor

function Mout= outputM(uout)
```

```
% Subfunction for producing the monitor function at every time.

% Produces a t_N by N matrix with rows corresponding to the
% monitor function evaluated at time t in tout.
Mout= zeros(t_N,N);
for i=1:t_N
    Mout(i,:)= monitor(uout(i,1:21)',uout(i,22:end)');
end % for i
end % function outputM

function utout= ut(y,yp)
% Subfunction for constructing the approximation to u_t

if nargin < 2
    xt_l= xt(y);
else                            % Implicit
    xt_l= yp(N+1:2*N);
end % if nargin
u= y(1:N);
x= y(N+1:2*N);

[C,F,B] = mk_fd_matrices;
u_xi= C*u;   % Central, forward and backward difference
x_xi= C*x;   % schemes for u and x.
u_xi_f= F*u;
u_xi_b= B*u;
x_xi_f= F*x;
x_xi_b= B*x;
tri111 = (diag(ones(N-1,1),-1) + eye(N) + diag(ones(N-1,1),1))/3;

utout= (u_xi./x_xi).*(xt_l - tri111*u) +...
     epsilon./(x_xi*h) .*...
     ((u_xi_f./x_xi_f) - (u_xi_b./x_xi_b));
utout(1)= 0; % Boundary Conditions
utout(N)= 0;
end % function ut

function xtout= xt(y)
% Subfunction for constructing approximation to x_t
% x_t = 1/tau*(x_xi*M(x))_xi;
u= y(1:N);
x= y(N+1:2*N);

% Forward/Backwards difference estimate of dx/d(xi).
x_xi= (x(2:end)-x(1:end-1))/h;
```

```
M_m= monitor(u, x);
M_b= ppval(basepp,x);
M= M_m.*M_b;

  % Linear interpolation for midpoints of M.
M_mid= (M(1:end-1)+M(2:end))*0.5;

xtout= 1/(tau*h) * [0;...
                (M_mid(2:end).*x_xi(2:end)...
                - M_mid(1:end-1).*x_xi(1:end-1));...
                0];
end % function xt

function myplot(uout, tout)
% function for plotting results
ts= [0,0.25,0.5,0.75,1]; % Values of t to plot.
gpoints=true; % Plot the coords as 'ro's?
clf
subplot(1,2,1)
hold on
for i = 1:length(ts)
    [~,t_i]= min(abs(tout-ts(i))); % tout closest to ts(i).
    t_i=t_i(1); % Incase we get two minima.
    plot(uout(t_i,N+1:2*N),uout(t_i,1:N),'b-')
    if gpoints
        plot(uout(t_i,N+1:2*N),uout(t_i,1:N),'ro')
    end
end % for i
xlim([0,1]), xlabel('x'), ylabel('u(x,t)')

subplot(1,2,2)
hold on
for i=N+1:2*N
    plot(uout(:,i),tout)
end % for i
xlim([0,1])
xlabel('x')
ylabel('t')
axis('square')
hold off
end % function myplot

end % Main function fd_adaptive
```

A.5 main.m

```
function [output] = main(Dt_in, T_in, N_in, alpha_q)
% A 1D vertical slice code with moisture
%
% Subfunctions:
%  - report
%  - calc_zD
%  - calc_R_D
%  - calc_L
%  - calc_R
%  - calc_H
%  - calc_RHS
%  - solve_help
%  - back_sub
%
% Removed for thesis
%  - mk_contour
%  - mk_plot
%
% Calls to external functions:
% ../mk_initial_conditions.m
%       Function which creates a specified initial condition, such as
%       isentropic or an inversion layer.
% ../moisture_simple.m
%       Moisture scheme which makes simplifications, such as assuming
%       c_{vml} is constant.
% ../moisture.m
%       As above, but without simplifications
% ../movemesh.m
%       Wrapper function for equidistribute.m
% ../interpolation/interp3lim.m
%       Cubic Lagrange interpolant with a flux limiter.
%
% [output] = main(Dt_in, T_in, N_in, alpha_q)

old_path = addpath([pwd,'\interpolation']);

global Dt T N Dz alpha alpha_X n1 is_moist
Dt = Dt_in;
T = T_in;
N = N_in;

%Dt = 60;                       % Timestep
```

```
%T = 300;                        % Number of timesteps
%N = 60;                        % Spatial resolution
z_ceiling = 10000;              % Domain ceiling
Dz = z_ceiling/N;               % Spatial step

alpha = 0.5;                    % SL weighting
alpha_X = 0.5;                  % SL weighting for d-points
n1 = 1;                         % Switch for theta correction

M = 4;                          % Departure point iterations
K = 4;                          % Helmholtz (Newton) iterations

is_moist = 0;                   % Moisture switch
do_move = 0;                    % Moving mesh switch
 move_param = 1;
heating = 0;                    % Gaussian heating switch

verbatim=0;                     % Level of verbose outputting

plotparam=2;                    % Type of plotting to do (see mk_plot)

% Constants
global cp g gamma R p0
cp = 1005;                      % Dry Specific heat at constant pressure
g =  9.80665;                   % Gravity
kappa = 0.285621890547264;      % R/cp
gamma = (1-kappa)/kappa;
R = 287.05;                     % Dry gas constant
p0 = 1e5;                       % Reference pressure



% Some initial mesh
[w,rho,theta,exner, z_A, zh_A] = mk_initial_conditions(alpha_q);
% Initialise the departure mesh to be equal to arrival mesh
z_D = z_A;
zh_D = zh_A;
% For a moving mesh; z_A_n means z_A at the previous timestep.
z_A_n = z_A;
zh_A_n = zh_A;

% Initialise moisture parameters
if is_moist
  rv = 0.005*ones(size(theta));
  rc = 0*rv;
else
```

```
 % theta = theta + 30/z_ceiling*z_A;
  theta(1) = theta(1) + 5.0*tanh(0/1800);
  rv = 0*ones(size(theta));
  rc = rv;
end
rv_n = rv;
rc_n = rc;
rvout = zeros(T,N+1);
rcout = zeros(T,N+1);

% Perturbation
%theta(5) = theta(5) + 5;

z_A0 = z_A;
zh_A0 = zh_A;
w0=w;
rho0=rho;
exner0=exner;
theta0=theta;
mass0 = sum(rho0.*diff(z_A0));
N_sq0 = g./theta0(2:end-1).*...
    (theta0(3:end)-theta0(1:end-2))./(z_A0(3:end)-z_A0(1:end-2));
N_sq0 = g*log(theta0(2:end)./theta0(1:end-1))./(z_A0(2:end)-z_A0(1:end-1));

w_n = w;
theta_n = theta;
rho_n = rho;
exner_n = exner;

for t=1:T % Timestep loop
% Calculate initial estimate of departure points
%z_D = z_A;
%zh_D = zh_A;
for m=1:M % Outer loop, departure point iteration
  % Calculate departure points from w^{(k)}_A and w^{n}_D:
  % Recalculate w_D, then the departure points again.
  z_D_old = z_D;
  [z_D,zh_D] = calc_zD(w,w_n,z_A,zh_A,z_D,zh_D);
  z_D_prime = z_D_old - z_D;
  % MM changed z_A to z_A_n
  [R_w_D,R_rho_D,R_theta_D] = ...
    calc_R_D(w_n,theta_n,rho_n,exner_n,rv,rc,z_A_n,z_D,zh_A_n,zh_D);
  for k=1:K % Inner loop. Newton iterations.
    [L_w,L_rho,L_theta,L_pi] = calc_L(w,rho,theta,exner,rv,rc,z_A,zh_A);
    [R_w,R_rho,R_theta,R_pi] = calc_R(L_w,L_rho,L_theta,L_pi,...
                                      R_w_D,R_rho_D,R_theta_D,exner);
```

```
    [H0,H1,H2,Htheta,Hw] = calc_H(w,rho,theta,exner,z_A,zh_A);
    [Rfrak_w,~, RHS] = calc_RHS(R_w,R_rho,R_theta,R_pi,...
                                rho,exner,theta,H0,H1,Hw,Htheta,z_A,zh_A);
    [exner_prime] = solve_helm(H0,H1,H2,Htheta,Hw,RHS,exner,theta,z_A,zh_A);

    [w,rho,theta,exner] = back_sub(w,rho,exner,theta,exner_prime,z_A,zh_A,...
                                   H0,H1,H2,Htheta,Hw,Rfrak_w,R_rho,R_theta,R_pi);
    theta(1) = 300 + 5.0*tanh(t*Dt/1800);
  end % for k, inner loop
end % for m, outer loop
% Store variables for next loop

if heating
  a=1; b=0.00001; z_i= 2/3*(N*Dz);
  delta_theta = a*exp(-b*(z_A - z_i).^2);
  theta = theta + delta_theta;
  %theta(floor(2/3*N)) = theta(floor(2/3*N))+4;
end % if heating

if is_moist
  rv_D = interp3lim(z_A_n,rv_n,z_D);
  rc_D = interp3lim(z_A_n,rc_n,z_D);
  theta_D = interp3lim(z_A_n,theta_n,z_D);

  switch 0 % 0 for simple thermodynamics, 1 for more complicated
    case 0
  %    [theta_out, rv_out, rc_out] = ...
      [theta, rv, rc] = ...
        moisture_simple(w, exner, theta, rv_D, rc_D, z_A, zh_A);
    case 1
      for mm=1:10
        %[r_dot_out, S_theta_out] = moisture(w, exner, theta, rv, rc, z_A, zh_A);
        [r_dot_out, S_theta_out] = moisture(w, exner, theta, rv_D, rc_D, z_A, zh_A);
        %rv = rv_D - Dt*r_dot_out([1,1:end,end]);
        %rc = rc_D + Dt*r_dot_out([1,1:end,end]);
        rv = rv_D - Dt*r_dot_out;
        rc = rc_D + Dt*r_dot_out;
        theta = theta_D + Dt*S_theta_out([1,1:end,end]);
      end % mm
  end % switch

  rv_n = rv;
  rc_n = rc;
  rvout(t,:) = rv;
  rcout(t,:) = rc;
end % if is_moist
```

```
w_n = w;
rho_n = rho;
theta_n = theta;
exner_n = exner;
% Save vars
if t==1
    wout = zeros(T,N+1);
    thetaout = zeros(T,N+1);
    zout = zeros(T,N+1);
    rhoout = zeros(T,N);
    exnerout = zeros(T,N);
    zhout = zeros(T,N);
    massout = zeros(T,1);

    Mout = zeros(T,N+1);
end


% MM
zout(t,:) = z_A;
zhout(t,:) = zh_A;

wout(t,:) = w;
rhoout(t,:) = rho;
thetaout(t,:) = theta;
exnerout(t,:) = exner;

massout(t) = sum(rhoout(t,:).*diff(zout(t,:)));

% MM section
% 1. These meshes become previous meshes
%
z_A_n = z_A;
zh_A_n = zh_A;
%
% 2. Move the mesh, either based on current fields or on predictions
%
if do_move
    if any(ismember(who,'move_param'))
      mm_alpha = move_param;
    else
      mm_alpha = 0.5;
    end % if ismember
    [z_A_adapted,Mout(t,:)] = movemesh(z_A,zh_A,theta);
    z_A = mm_alpha * z_A_n + (1-mm_alpha)*z_A_adapted;
```

```
    zh_A = (z_A(1:end-1) + z_A(2:end))/2;
end
%
% 3. Interpolate w, rho, theta and exner (currently intial guesses
%    for next timestep) from z_A_n onto z_A
theta = interp3lim(z_A_n,theta_n,z_A);
w = interp3lim(z_A_n,w_n,z_A);
exner = interp3lim(zh_A_n,exner_n,zh_A);
rho = interp3lim(zh_A_n,rho_n,zh_A);


end % for t, timestep loop

for t=1:T % Calculate the deviations from the (interpolated) initial fields
  thetadev(t,:) = thetaout(t,:)-interp3lim(z_A0,theta0,zout(t,:));
  rhodev(t,:) = (rhoout(t,:)-interp3lim(zh_A0,rho0,zhout(t,:)))...
    ./interp3lim(zh_A0,rho0,zhout(t,:));
  exnerdev(t,:) = exnerout(t,:)-interp3lim(zh_A0,exner0,zhout(t,:));
end % for t, deviations
massdev = (massout-mass0)/mass0;
massdiff = diff(massout)./massout(1:end-1);

%mk_plot(plotparam, wout, rhodev, thetadev, exnerdev, ...
%      z_A, zh_A, zout, massdev, rvout, rcout, z_A0, massdiff)

path(old_path);

% Plot some data
figure;
plot(theta0,z_A,'k',theta,z_A,'b','linewidth',2);
end % function main

function [z_D,zh_D] = calc_zD(w,w_n,z_A,zh_A,z_D,zh_D)
global Dt alpha_X
zh_weighted_ave = 1;
zh_interp = 2;

for ii = 1:2 % Departure point iteration
  w_D = interp3lim(z_A,w_n,z_D);
  z_D = z_A - Dt*(alpha_X*w + (1-alpha_X)*w_D);
end % for i
z_D = min(max(z_A),max(z_D,0));
switch(zh_interp)
  case(zh_weighted_ave)
    zh_D = (z_D(1:end-1) + z_D(2:end))/2;
  case(zh_interp)
```

```
    w_h = 0.5*(w(2:end) + w(1:end-1));
    for ii = 1:2 % Departure point iteration
      w_Dh = interp3lim(z_A,w_n,zh_D);
      zh_D = zh_A - Dt*(alpha_X*w_h + (1-alpha_X)*w_Dh);
    end % for i
  otherwise
    error 'Unknown type of z_h interoplation'
end % switch
zh_D = min(max(z_A),max(zh_D,0));
end % function calc_zD

function [R_w_D, R_rho_D, R_theta_D] = ...
    calc_R_D(w_n, theta_n, rho_n, exner_n, rv, rc, z_A_n, z_D, zh_A_n, zh_D)
% Calculates R_w^n, R_\rho^n and R_\theta^n on the previous meshes z_A,
% zh_A and interpolates onto the departure points z_D, zh_D.

% Uses the interpolation routine interp3lim (currently in the folder
% ~/dos/MATLAB/ ).
global Dt alpha cp g
% could possibly calculate Dz_A before-hand, but doing it internally
% will closer fit the moving mesh approach.
% This is the arrival mesh from the previous time step.
Dz_A = diff(z_A_n);
Dzh_A = diff(zh_A_n);

exner_z_bot = -g*(1+rv(1) + rc(1))/(cp*theta_n(1));
exner_z_internal = (exner_n(2:end) - exner_n(1:end-1))./Dzh_A;
exner_z_top = -g*(1+rv(end) + rc(end))/(cp*theta_n(end));
% This is equivalent to saying exner_z in linear in [z_A(0), z_D(2)] and
% [z_D(end-1), z_A(end)]. (interval-and-a-half)
exner_z = [exner_z_bot,exner_z_internal, exner_z_top];
w_z = (w_n(2:end) - w_n(1:end-1))./Dz_A;

R_n_w = w_n - Dt*(1-alpha)*(cp./(1+rv+rc).*theta_n.*exner_z + g);
R_n_rho = rho_n - Dt*(1-alpha)*rho_n.*w_z;

R_w_D = interp3lim(z_A_n,R_n_w,z_D);
R_theta_D=interp3lim(z_A_n, theta_n, z_D);
R_rho_D=interp3lim(zh_A_n, R_n_rho, zh_D);

end % function calc_R_D

function [L_w,L_rho,L_theta,L_pi] = calc_L(w,rho,theta,exner,rv,rc,z_A,zh_A)
global Dt alpha cp g gamma R p0 % Dz
% The arrival mesh for current time
Dz_A = diff(z_A);
```

```
Dzh_A = diff(zh_A);

exner_z_bot = -g*(1+rv(1)+rc(1))/(cp*theta(1));
exner_z_internal = (exner(2:end) - exner(1:end-1))./Dzh_A;
exner_z_top = -g*(1+rv(end)+rc(end))/(cp*theta(end));

exner_z = [exner_z_bot, exner_z_internal, exner_z_top];
w_z = (w(2:end) - w(1:end-1))./Dz_A;
ave_weight = (zh_A - z_A(1:end-1))./(z_A(2:end)-z_A(1:end-1));
theta_ave = (theta(1:end-1).*ave_weight + theta(2:end).*(1-ave_weight));

L_w = w + Dt*alpha*(cp./(1+rv+rc).*theta.*exner_z + g);
L_rho = rho + Dt*alpha*rho.*w_z;
L_theta = theta;
L_pi = (exner.^gamma - R/p0*rho.*theta_ave);
end % function calc_L

function [R_w,R_rho,R_theta,R_pi] = calc_R(L_w,L_rho,L_theta,L_pi,...
                                    R_w_D,R_rho_D,R_theta_D,exner)
global gamma
R_w = R_w_D - L_w;
R_rho = R_rho_D - L_rho;
R_theta = R_theta_D - L_theta;
R_pi = -L_pi./(exner.^gamma);
end % function calc_R

function [H0,H1,H2,Htheta,Hw] = calc_H(w,rho,theta,exner,z_A,zh_A)
global alpha n1 Dt cp R p0 gamma % Dz
Dz_A = diff(z_A);
Dzh_A = diff(zh_A);
w_z = (w(2:end)-w(1:end-1))./Dz_A;

ave_weight = (zh_A - z_A(1:end-1))./(z_A(2:end)-z_A(1:end-1));
theta_ave = (theta(1:end-1).*ave_weight + theta(2:end).*(1-ave_weight));
theta_z_int = (theta(3:end) - theta(1:end-2))./(Dz_A(1:end-1)+Dz_A(2:end));
theta_z = [0, theta_z_int, 0];
exner_z_int = (exner(2:end) - exner(1:end-1))./Dzh_A;
exner_z = [0,exner_z_int,0];

H0 = R/p0*(rho.*theta_ave./(exner.^gamma));
H1 = 1./(1+alpha*Dt*w_z);
H2 = alpha*Dt*cp*theta;
Htheta = n1*alpha*Dt*theta_z;
Hw = 1./(1-alpha*Dt*cp*Htheta.*exner_z);
end % function calc_H
```

```
function [Rfrak_w, Rfrak_pi, RHS] = ...
  calc_RHS(R_w,R_rho,R_theta,R_pi,rho,exner,theta,H0,H1,Hw,Htheta,z_A,zh_A)
global alpha Dt cp % Dz
Dz_A = diff(z_A);
Dzh_A = diff(zh_A);

exner_z_internal = (exner(2:end) - exner(1:end-1))./Dzh_A;
exner_z = [0,exner_z_internal,0];
ave_weight = (zh_A - z_A(1:end-1))./(z_A(2:end)-z_A(1:end-1));
R_tt_ave = (R_theta(1:end-1)./theta(1:end-1).*ave_weight +...
                  R_theta(2:end)./theta(2:end).*(1-ave_weight));

Rfrak_w = R_w - alpha * Dt * cp *R_theta.*exner_z;
Rfrak_pi = R_pi + H0.*H1.*(R_rho./rho) + H0.*R_tt_ave;

Hw_Rfrak_w_z = (Hw(2:end).*Rfrak_w(2:end) - ...
                Hw(1:end-1).*Rfrak_w(1:end-1))./Dz_A;
HHR = Htheta.*Hw.*Rfrak_w./theta;
HHR_ave = ave_weight.*HHR(1:end-1) + (1-ave_weight).*HHR(2:end);

RHS = Rfrak_pi - alpha*Dt*H0.*H1.*Hw_Rfrak_w_z - H0.*HHR_ave;
end % function calc_RHS

function [exner_prime] = solve_helm(H0,H1,H2,Htheta,Hw,RHS,exner,theta,z_A,zh_A)
global gamma alpha Dt % Dz
Dz_A = diff(z_A);
Dzh_A = diff(zh_A);
ave_weight = (zh_A - z_A(1:end-1))./(z_A(2:end)-z_A(1:end-1));

N = length(RHS);
A_diag = zeros(1,N);
A_sup = zeros(1,N);
A_sub = zeros(1,N);

K0 = alpha*Dt*H0.*H1./Dz_A;
K1 = Hw.*H2./[1,Dzh_A,1];
K1(1)=0; K1(end) = 0;
K2 = Htheta.*Hw.*H2./(theta.*[1,Dzh_A,1]);
K2(1)=0; K2(end) = 0;

A_diag = gamma./exner - (-K0.*(K1(2:end)+K1(1:end-1)) + ...
    H0.*(ave_weight.*K2(1:end-1) - (1-ave_weight).*K2(2:end)));
A_sup = -(K0.*K1(2:end) + H0.*(1-ave_weight).*K2(2:end));
A_sub = -(K0.*K1(1:end-1) - H0.*ave_weight.*K2(1:end-1));

% BCs
```

```
A_diag(1) = gamma/exner(1) - (-K0(1)*K1(2) + ...
    H0(1)*(-1)*(1-ave_weight(1))*K2(2));
A_diag(end) = gamma/exner(end) - (-K0(end).*K1(end-1) + ...
    H0(end).*ave_weight(end).*K2(end-1));

% Could use the Thomas algorithm
A = diag(A_diag) + diag(A_sup(1:end-1),1) + diag(A_sub(2:end),-1);
exner_prime = (A\(RHS'))';
end % function solve_helm

function [w,rho,theta,exner] = back_sub(w,rho,exner,theta,exner_prime,z_A,zh_A,...
                                        H0,H1,H2,Htheta,Hw,Rfrak_w,R_rho,R_theta,R_pi)
global gamma Dt
Dzh_A = diff(zh_A);
exner_prime_z_internal = (exner_prime(2:end)-exner_prime(1:end-1))./Dzh_A;
% BCs used here
exner_prime_z = [0,exner_prime_z_internal,0];

w_prime = Hw.*(Rfrak_w - H2.*exner_prime_z);
w_prime(end) = 0;

theta_prime = R_theta - Htheta.*w_prime;
theta_theta_ave = (theta_prime(1:end-1)./theta(1:end-1) + ...
                   theta_prime(2:end)./theta(2:end))/2;

% Choices for rho definition
with_R_rho = 1;
with_R_pi = 2;
with_sl_pi = 3;
rho_opt = 2;
switch(rho_opt)
  case(with_R_rho)
    Dz_A = diff(z_A);
    w_prime_z = (w_prime(2:end)-w_prime(1:end-1))./Dz_A;
    rho_prime = H1.*(R_rho - alpha*Dt* rho.*w_prime_z);
  case(with_R_pi)
    rho_prime = rho./H0.*(gamma*exner_prime./exner - R_pi) - ...
        rho.*theta_theta_ave;
  case(with_sl_pi)
end % switch rho_opt
exner = exner + exner_prime;
w = w + w_prime;
theta = theta + theta_prime;
rho = rho + rho_prime;
end % function back_sub
```

Appendix B

Derivatives of Burgers' equation

B.1 Time-Derivatives

Some equations from differentiating Burgers' equation (3.1):

$$u_{tx} = -(u_x)^2 - uu_{xx} + \varepsilon u_{3x}\,, \tag{B.1}$$

$$u_{txx} = -uu_{3x} - 3u_x u_{xx} + \varepsilon u_{4x}\,, \tag{B.2}$$

$$u_{tt} = u^2 u_{xx} - 2\varepsilon(uu_{3x} + u_x u_{xx}) + \varepsilon^2 u_{4x}\,. \tag{B.3}$$

The travelling wave solution (3.4) has the following spatial derivatives:

$$u_x = \frac{1}{2\varepsilon}\left((c-u)^2 - \alpha^2\right)\,, \tag{B.4}$$

$$u_{xx} = \frac{1}{2\varepsilon^2}\left(\alpha^2(c-u) - (c-u)^3\right)\,, \tag{B.5}$$

$$u_{3x} = -\frac{1}{\varepsilon}\left(u_{xx}(c-u) - (u_x)^2\right)\,, \tag{B.6}$$

$$u_{4x} = -\frac{1}{\varepsilon}\left(u_{3x}(c-u) - u_x u_{xx} - 2u_x u_{xx}\right)\,. \tag{B.7}$$

These can be expressed in polynomials of u but are considerably less compact.

B.2 Integrals of tanh

For

$$v(x) = c - \alpha\tanh\left(\frac{\alpha}{2\varepsilon}x\right)\,, \tag{B.8}$$

we can evaluate the following integrals over the real line:

$$\int_{-\infty}^{\infty} v^2 v'' \,\mathrm{d}x = \frac{-4}{3}\frac{\alpha^3 c}{\varepsilon}\,. \tag{B.9}$$

$$\int_{-\infty}^{\infty} v v'' \,\mathrm{d}x = \frac{-2}{3}\frac{\alpha^3}{\varepsilon}\,. \tag{B.10}$$

$$\int_{-\infty}^{\infty} v v'^2 \,\mathrm{d}x = \frac{2}{3}\frac{\alpha^3 c}{\varepsilon}\,. \tag{B.11}$$

$$\int_{-\infty}^{\infty} v'^2 \,\mathrm{d}x = \frac{2}{3}\frac{\alpha^3}{\varepsilon}\,. \tag{B.12}$$

List of Figures

Bibliography

[1] J. Behrens, *Adaptive atmospheric modeling: key techniques in grid generation, data structures, and numerical operations with applications*, Springer, 2006.

[2] R. Bermejo and A. Staniforth, *The conversion of semi-Lagrangian advection schemes to quasi-monotone schemes*, Monthly Weather Review **120** (1992), no. 11, 2622–2632.

[3] G.H. Bryan and J.M. Fritsch, *A benchmark simulation for moist nonhydrostatic numerical models*, Monthly Weather Review **130** (2002), no. 12, 2917–2928.

[4] C.J. Budd, M.J.P. Cullen, and E.J. Walsh, *Monge-Ampére based moving mesh methods for numerical weather prediction, with applications to the Eady problem*, Journal of Computational Physics **236** (2013), 247–270.

[5] C.J. Budd, W. Huang, and R.D. Russell, *Adaptivity with moving grids*, Acta Numerica **18** (2009), no. 1, 111–241.

[6] C.J. Budd and J.F. Williams, *Moving mesh generation using the parabolic Monge-Ampère equation*, SIAM Journal on Scientific Computing **31** (2009), no. 5, 3438–3465.

[7] E. Cordero, N. Wood, and A. Staniforth, *Impact of semi-Lagrangian trajectories on the discrete normal modes of a non-hydrostatic vertical-column model*, Quarterly Journal of the Royal Meteorological Society **131** (2005), no. 605, 93–108.

[8] J. Côté, S. Gravel, A. Méthot, A. Patoine, M. Roch, and A. Staniforth, *The operational CMC-MRB global environmental multiscale (GEM) model. part I: Design considerations and formulation*, Monthly Weather Review **126** (1998), 1373–1395.

[9] M.J.P. Cullen, *Alternative implementations of the semi-Lagrangian semi-implicit schemes in the ECMWF model*, Quarterly Journal of the Royal Meteorological Society **127** (2001), no. 578, 2787–2802.

[10] T. Davies, M.J.P. Cullen, A.J. Malcolm, M.H. Mawson, A. Staniforth, A.A. White, and N. Wood, *A new dynamical core for the Met Office's global and regional modelling of the atmosphere*, Quarterly Journal of the Royal Meteorological Society **131** (2005), no. 608, 1759–1782.

[11] P.J. Davis, *Interpolation and approximation*, Dover Books on Mathematics, Blaisdell Publishing, 1963.

[12] D.R. Durran, *Numerical methods for fluid dynamics: With applications to geophysics*, vol. 32, Springer Science & Business Media, 2010.

[13] L.C. Evans, *Partial differential equations*, second ed., Graduate studies in mathematics, American Mathematical Society, 2010.

[14] R. Ford, C.C. Pain, M.D. Piggott, A.J.H. Goddard, C.R.E. De Oliveira, and A.P. Umpleby, *A nonhydrostatic finite-element model for three-dimensional stratified oceanic flows. part I: Model formulation*, Monthly Weather Review **132** (2004), no. 12, 2816–2831.

[15] F.N. Fritsch and R.E. Carlson, *Monotone piecewise cubic interpolation*, SIAM Journal on Numerical Analysis **17** (1980), no. 2, 238–246.

[16] C.I. Hampson, *Estimates of mesh quality in grids generated for aerospace applications*, Master's thesis, University of Bath, UK, 2011.

[17] A. Harten, B. Engquist, S. Osher, and S.R. Chakravarthy, *Uniformly high order accurate essentially non-oscillatory schemes, III*, Journal of Computational Physics **71** (1987), no. 2, 231–303.

[18] J.R. Holton and G.J. Hakim, *An introduction to dynamic meteorology*, fifth ed., Academic Press, 2012.

[19] W. Huang and R.D. Ren, Y. Russell, *Moving mesh partial differential equations (MMPDES) based on the equidistribution principle*, SIAM Journal on Numerical Analysis **31** (1994), no. 3, 709–730.

[20] W. Huang and R.D. Russell, *Analysis of moving mesh partial differential equations with spatial smoothing*, SIAM Journal on Numerical Analysis **34** (1997), no. 3, 1106–1126.

[21] ———, *Adaptive moving mesh methods*, Applied Mathematical Sciences, Springer, 2010.

[22] J.M. Hyman, *Accurate monotonicity preserving cubic interpolation*, SIAM Journal on Scientific and Statistical Computing **4** (1983), no. 4, 645–654.

[23] C. Kühnlein, P.K. Smolarkiewicz, and A. Dörnbrack, *Modelling atmospheric flows with adaptive moving meshes*, Journal of Computational Physics **231** (2012), no. 7, 2741–2763.

[24] Hung-Chi Kuo and R.T. Williams, *Semi-Lagrangian solutions to the inviscid Burgers equation*, Monthly Weather Review **118** (1990), no. 6, 1278–1288.

[25] J. Lang, W. Cao, W. Huang, and R.D. Russell, *A two-dimensional moving finite element method with local refinement based on a posteriori error estimates*, Applied Numerical Mathematics **46** (2003), no. 1, 75–94.

[26] P.H. Lauritzen, E. Kaas, and B. Machenhauer, *A mass-conservative semi-implicit semi-Lagrangian limited-area shallow-water model on the sphere*, Monthly Weather Review **134** (2006), no. 4, 1205–1221.

[27] P.H. Lauritzen, E. Kaas, B. Machenhauer, and K. Lindberg, *A mass-conservative version of the semi-implicit semi-Lagrangian HIRLAM*, Quarterly Journal of the Royal Meteorological Society **134** (2008), no. 635, 1583–1595.

[28] D.Y. Le Roux, C.A. Lin, and A. Staniforth, *A semi-implicit semi-Lagrangian finite-element shallow-water ocean model*, Monthly Weather Review **128** (2000), no. 5, 1384–1401.

[29] R.J. LeVeque, *Numerical methods for conservation laws*, Lectures in mathematics ETH Zürich, Springer, 1992.

[30] ———, *Finite volume methods for hyperbolic problems*, vol. 31, Cambridge University Press, 2002.

[31] B. Machenhauer, E. Kaas, and P.H. Lauritzen, *Finite-volume methods in meteorology*, Handbook of Numerical Analysis **14** (2009), 3–120.

[32] T. Melvin, M. Dubal, N. Wood, A. Staniforth, and M. Zerroukat, *An inherently mass-conserving iterative semi-implicit semi-Lagrangian discretization of the non-hydrostatic vertical-slice equations*, Quarterly Journal of the Royal Meteorological Society **136** (2010), no. 648, 799–814.

[33] K.W. Morton and D.F. Mayers, *Numerical solutions of partial differential equations: An introduction*, Cambridge University Press, 1994.

[34] C. Piccolo and M.J.P. Cullen, *Adaptive mesh method in the Met Office variational data assimilation system*, Quarterly Journal of the Royal Meteorological Society **137** (2011), no. 656, 631–640.

[35] W.H. Press, *Numerical recipes 3rd edition: The art of scientific computing*, Cambridge University Press, 2007.

[36] A. Priestley, *A quasi-conservative version of the semi-Lagrangian advection scheme*, Monthly Weather Review **121** (1993), no. 2, 621–629.

[37] P.J. Rasch and D.L. Williamson, *On shape-preserving interpolation and semi-Lagrangian transport*, SIAM Journal on Scientific and Statistical Computing **11** (1990), no. 4, 656–687.

[38] R.W. Riddaway, *Course notes from ECMWF numerical methods training course*, 2001, Available at `http://www.ecmwf.int/newsevents/training/rcourse_notes/NUMERICAL_METHODS/NUMERICAL_METHODS/index.html`, retrieved 01/09/2013.

[39] H. Ritchie, *Eliminating the interpolation associated with the semi-Lagrangian scheme*, Monthly Weather Review **114** (1986), no. 1, 135–146.

[40] H. Ritchie, C. Temperton, A. Simmons, M. Hortal, T. Davies, D. Dent, and M. Hamrud, *Implementation of the semi-Lagrangian method in a high resolution version of the ECMWF forecast model*, Monthly Weather Review **123** (1995), 489–514.

[41] C. Rivest, A. Staniforth, and A. Robert, *Spurious resonant response of semi-Lagrangian discretizations to orographic forcing: Diagnosis and solution*, Monthly Weather Review **122** (1994), no. 2, 366–376.

[42] A. Robert, *A stable numerical integration scheme for the primitive meteorological equations*, Atmosphere-Ocean **19** (1981), 35–46.

[43] ———, *A semi-Lagrangian and semi-implicit numerical integration scheme for the primitive meteorological equations*, Journal of the Meteorological Society of Japan **60** (1982), 319–325.

[44] J. Robinson, *Quadrilateral and hexahedron shape parameters*, Finite elements in analysis and design **16** (1994), no. 1, 43–52.

[45] J.S. Sawyer, *A semi-Lagrangian method of solving the vorticity advection equation*, Tellus **15** (1963), no. 4, 336–342.

[46] C. Schwab, *p-and hp-finite element methods: Theory and applications in solid and fluid mechanics*, Oxford University Press, 1998.

[47] C.J. Smith, *The semi-Lagrangian method in atmospheric modelling*, Ph.D. thesis, University of Reading, 2000.

[48] P.K. Smolarkiewicz and G.A. Grell, *A class of monotone interpolation schemes*, Journal of Computational Physics **101** (1992), no. 2, 431–440.

[49] A. Staniforth and J. Côté, *Semi-Lagrangian integration schemes for atmospheric models - a review*, Monthly Weather Review **119** (1991), no. 9, 2206–2223.

[50] Huazhong Tang and Tao Tang, *Adaptive mesh methods for one-and two-dimensional hyperbolic conservation laws*, SIAM Journal on Numerical Analysis **41** (2003), no. 2, 487–515.

[51] C. Temperton, M. Hortal, and A. Simmons, *A two-time-level semi-Lagrangian global spectral model*, Quarterly Journal of the Royal Meteorological Society **127** (2001), no. 571, 111–127.

[52] C. Temperton and A. Staniforth, *An efficient two-time-level semi-Lagrangian semi-implicit integration scheme*, Quarterly Journal of the Royal Meteorological Society **113** (1987), no. 477, 1025–1039.

[53] P.A. Ullrich, C. Jablonowski, J. Kent, P.H. Lauritzen, R.D. Nair, and M.A. Taylor, *Dynamical core model intercomparison project (DCMIP) test case document, version 1.7*, `https://www.earthsystemcog.org/projects/dcmip-2012/test_cases`, retrieved 20/08/2015, 2012.

[54] A. Wiin-Nielsen, *On the application of trajectory methods in numerical forecasting*, Tellus **11** (1959), no. 2, 180–196.

[55] D.L. Williamson and P.J. Rasch, *Two-dimensional semi-Lagrangian transport with shape-preserving interpolation*, Monthly Weather Review **117** (1989), no. 1, 102–129.

[56] N. Wood, A. Staniforth, A. White, T. Allen, M. Diamantakis, M. Gross, T. Melvin, C. Smith, S. Vosper, M. Zerroukat, et al., *An inherently mass-conserving semi-implicit semi-Lagrangian discretization of the deep-atmosphere global non-hydrostatic equations*, Quarterly Journal of the Royal Meteorological Society **140** (2014), no. 682, 1505–1520.

[57] N. Wood, A. Staniforth, A. White, J. Thuburn, T. Allen, T. Davies, M. Diamantakis, M. Dubal, M. Gross, T. Melvin, C. Smith, and M. Zerroukat, *ENDGame formulation v3.02*, Tech. report, Met Office, Exeter, UK, 2011.

[58] N. Wood, A.A. White, and A. Staniforth, *Treatment of vector equations in deep-atmosphere, semi-Lagrangian models. II: Kinematic equation*, Quarterly Journal of the Royal Meteorological Society **136** (2010), no. 647, 507–516.

[59] Kao-San Yeh, J. Côté, S. Gravel, A. Méthot, A. Patoine, M. Roch, and A. Staniforth, *The CMC-MRB global environmental multiscale (GEM) model. part III: Nonhydrostatic formulation*, Monthly Weather Review **130** (2002), no. 2, 339–356.

[60] M. Zerroukat, N. Wood, A. Staniforth, A.A. White, and J. Thuburn, *An inherently mass-conserving semi-implicit semi-Lagrangian discretisation of the shallow-water equations on the sphere*, Quarterly Journal of the Royal Meteorological Society **135** (2009), no. 642, 1104–1116.

Index